Date: February 20, 2025

First Project Meeting and Idea Discussion

What did we do today?
Today, we had our first project meeting. We spent time chatting and getting to know each other's strengths, which was important for task distribution.

Meeting Summary:
- Everyone shared different project ideas.
- We discussed the examples provided in class and analyzed them together.
- By the end of the meeting, we reached a common decision on how our project would be structured.
- Each team member was assigned the task of conducting general research on the project.

Future Plans:
- We scheduled our next meeting for **Tuesday, February 25, at 10:00.**
- In this meeting, we will share the results of our research and distribute specific tasks accordingly.

Date: February 25, 2025

Project Meeting and Task Distribution

What did we do Today?
Today, we held a meeting with our project team on Discord. The main purpose of the meeting was to share the research we had done, clarify task distribution, and discuss how to best prepare for the first milestone.

Meeting Summary:
- Everyone presented their research on the project, and we evaluated the gathered information.
- We assigned specific tasks to each team member.
- We created a plan outlining the steps needed to complete the first milestone.

Challenges We Faced:
- A change needed to be made regarding the game characters. Everyone had different ideas, but by the end of the meeting, we reached a common agreement on how the characters should be designed.

Future Plans:
- Everyone will focus on completing their tasks until the next meeting.
- Our next meeting is scheduled for **Thursday, February 27 at 12:00.** In the meeting, we will share our progress with each other.

Date: February 27, 2025

Project Meeting and preparation for the presentation

What did we do today?
Today, we discussed our Game name and decided to name our game: **Think Outside the room**.
We talked about our completed game concept and worked on our presentation for the next week.

Meeting summary:
- We decided our game name: **Think Outside The Room**
- We assign who is responsible for what in the presentation.

Future Plans:
Everyone will work on the presentation

Date: March 7, 2025

Project Meeting and preparation for the presentation

What did we do today?
Today, we held a meeting to discuss Milestone 2 and distribute tasks among the project team for it.

Meeting summary:
- We talked about Milestone 2 and how to prepare for it more effectively.
- Task distribution was finalized for the team, ensuring everyone has a clear role.
- We discussed strategies to improve our workflow and meet the milestone requirements efficiently

Future Plans:
- Team members will begin working on their assigned tasks.
- .A follow-up meeting will be scheduled to discuss the challenges if necessary.

Date: March 17, 2025

Chat Feature Testing and Command Standardization

What did we do today?
Today, we tested new functionalities for our game and improved our project by working collaboratively, ensuring smooth integration and quality time.

Meeting summary:
- Aisha coded the **chat feature**, and we tested it together to ensure smooth functionality.

- Illia created a standard for sending commands to the server, defining how the server processes and handles these commands. He explained the implementation with examples.
- Using this new standard, we successfully integrated the **nickname changing** feature, allowing players to change their nicknames through the server.

Future Plans:
- Continue refining the chat system and ensure commands are properly handled by the server.
- Plan another session to test and validate more **server-side** functionalities.

Date: March 20, 2025

Project Meeting and preparation for the presentation

What did we do today?
We merged the **login/logout branches** and improved the login functionality in our game.

Meeting summary:
- The **login system** was implemented to create a player and enable keybinds without assigning a random player.
- The following **server commands** have been tested and are now functional:
  create -> Creates player
  login -> Assigns an ID to the created player object
  logout -> Deletes the player from the session.
  exit -> Terminates the client.

Future Plans:
- Plan another session to review and test additional command functionalities.

Date: April 2, 2025

Task Distribution for Milestone 3

What did we do today?
Today, we held a team meeting to plan for Milestone 3. During the meeting, we discussed the main components of the milestone and assigned responsibilities across the team.

Meeting summary:
- We went over the different aspects of Milestone 3, including development, testing, documentation, and presentation.
- Tasks were distributed based on individual strengths and previous contributions

- We made sure that every team member is involved in at least one area, and most responsibilities are shared to encourage collaboration.
- Special attention was given to key areas such as game logic, GUI, protocol, login and lounge systems, as well as the final presentation.

Future Plans:
- Everyone will begin working on their assigned parts of Milestone 3.
- Open questions (like a few undefined features) will be clarified in upcoming meetings.
- We'll continue to collaborate closely and track progress to stay on schedule.

Date: April 3, 2025
Author: William

Argument Parsing to the Main class

What did we do today?
- Today I implemented a command-line argument parser so that users can specify the server/client mode, IP, and port when launching the game.
- I introduced a Rope mechanic into the main class with reverse kinematics, laying groundwork for the item rope.

What did I learn?
- how to do reverse kinematics and how to work with argument parser.

Challenges
- Ensuring the argument parsing did not conflict with JavaFX startup.
- Integrating a Rope concept without fully defined physicsä
- Making sure that the JavaFX environment initializes in the correct order, without Exceptions.

Future Plans
- Synchronize the rope

Date: April 4, 2025
Author: Sena

GUI Implementation for Chat and Lobby Features

What did we do today?
Today Illia and I worked together on implementing several key features into the game's graphical user interface (GUI). These features are essential for enhancing the multiplayer experience and communication within the game.

Meeting summary:
- We designed and integrated a multi-lobby system, allowing each game to have its own dedicated lobby along with a separate internal chat.

- We implemented a way to list players within each lobby, so users can see who they are playing with.
- Additionally, we developed the Whisper, Global, and Lobby Chat tabs as part of the chat system UI.
- While the features are not fully functional yet in terms of backend communication, we successfully built a clear, user-friendly interface that is ready to be connected with the underlying server logic.

Challenges We Faced:
- Since the server-side logic for these features was still under development, we had to simulate behavior and make assumptions about how data would flow between the client and server.
- Ensuring a consistent user experience across different chat modes also required several design iterations.

What did I learn?
- Collaborating on UI design helps align both technical functionality and user experience.
- Even if features are not yet complete, building a solid visual structure makes the next development steps more efficient.

Future Plans:
- Connect the implemented UI elements to the actual server-side logic.
- Finalize and test the chat functionalities across all lobby types.
- Continue working collaboratively to polish and integrate remaining components.

Date: April 4, 2025
Author: Aiysha

What I did today:
I experimented with several physics libraries (jBox2D, dyn4j) to handle collision detection and gravity. Due to persistent issues with JavaFX module not found errors, I ultimately decided to implement my own collision detection and gravity system.

To support this, I restructured the game objects by creating custom classes for players, moving platforms, and static objects. I also updated the collision resolution logic so that static objects (like platforms) aren't pushed around by players. Additionally, I introduced modularity by adding flags (e.g., movable vs. immovable) to allow different game objects to behave differently in terms of movement and collisions.

Challenges I faced:

- **JavaFX module issues:** I encountered errors related to missing JavaFX runtime components, which made it difficult to use external physics libraries.
- **Gradle and module-path configuration problems:** These made integrating external libraries more complicated.

- **Interpolation vs. performance:** Balancing smooth interpolation for moving platforms with the need for efficient, lag-free updates was tricky.

- **Custom collision handling:** Ensuring that my custom system correctly handled different object types (players, platforms, boxes) with varying movement and mass properties took careful design and testing.
-

What did I learn:

- How to design a custom collision detection and resolution system tailored to the specific needs of our game.
- Techniques for smooth motion using interpolation (cosine interpolation) to make moving platforms appear fluid.

Future plans:

- Implement gravity based on object mass to give players and boxes more realistic behavior.

- Refine the collision resolution further, ensuring that objects like platforms remain static when they should.

- Add mechanics for grabbing and throwing objects, allowing for more interactive gameplay.

- Explore integrating additional game features (e.g., power-ups, enemy AI) once the core physics and collision systems are stable.

- Continue improving the structure and modularity of game objects for easier maintenance and future expansion.

Date: April 5, 2025
Author: Aiysha

What I Did Today

- Reviewed and reworked our player physics system using vector-based integration (acceleration → velocity → position).
  Experimented with implementing gravity and collision detection for the player locally.
- Investigated issues with jump mechanics, particularly the conditions that enable a jump (using onGround and canJump flags).
- Addressed various synchronization challenges between client and server physics updates.
- Ultimately decided to implement our own collision detection and gravity system to avoid JavaFX module not found errors.

Challenges I Faced

- Integrating physics using vector math and ensuring the correct update order (acceleration, then velocity, then position).
- Handling input correctly so that jump impulses weren't immediately canceled or interfered with by collision detection.
- Dealing with synchronization issues between client-side and server-side updates.
- Resolving the JavaFX module not found error, which hindered our ability to use built-in JavaFX physics.

What Did I Learn

- How to implement basic physics integration (using acceleration to update velocity and then position).
- The importance of maintaining proper state flags (like onGround and canJump) for accurate jump detection.
- Strategies to troubleshoot and resolve synchronization issues in a networked game environment.
- How to use external libraries (e.g., dyn4j's Vector2) for vector math, even though we eventually opted for our own implementation due to module issues.

Future Plans

- Refine our custom collision detection and gravity system to further reduce lag and jitter.
  Optimize the input handling mechanism to improve the responsiveness of player movements and jumps.
- Test extensively across various game scenarios to ensure consistent behavior.
- Investigate integrating more advanced physics libraries once the JavaFX module issues are resolved or a workaround is found.

Date: April 5, 2025
Author: Aiysha

What I Did Today:

- Box Synchronisation:
  Implemented a system to synchronise the state of boxes across clients using client authority. This ensures that box positions, movements, and interactions remain consistent for all players.
- Grabbing Boxes:
  Developed the functionality to allow players to grab boxes, making them follow the player's movements. This included logic to detect when a box is picked up and to update its state accordingly.
- Throwing Boxes:
  Added mechanics for throwing boxes. This feature calculates and applies throw velocity and direction so that boxes behave realistically when thrown.
- Client Authority Based Synchronisation:
  Adopted a client authority approach where the client controlling a box is responsible for its state updates. This helped in reducing server load and making

the interactions feel more responsive.

Challenges I Faced:

- Ensuring that synchronisation of box states was both consistent and low-latency across different clients.
- Handling edge cases when multiple players interacted with the same box, ensuring that the client authority model prevented conflicts.
- Fine-tuning the physics for grabbing and throwing to create a realistic and enjoyable experience.

What I Learned:

- How to implement client-side authority to handle real-time object synchronisation in a multiplayer game.
- The importance of state management when multiple clients can interact with the same game object.
- Techniques for implementing and debugging interactive gameplay mechanics like grabbing and throwing objects.
- Strategies to ensure smooth and consistent user experience in a networked environment.

Future Plans:

- Refinement of Physics:
  Continue to refine the physics behind box interactions, including edge cases and potential conflict resolution when multiple clients interact.
- Extend Gameplay Mechanics:
  Explore additional mechanics such as stacking boxes, collisions, and interactions with other game elements.
- User Feedback & Testing:
  Gather feedback from playtesting sessions to further tweak and optimise the synchronisation and interaction systems.

Date: April 5, 2025

Author: William

What I Did Today:

- debugging mainclass to ensure UI Launches successfully.
- Fixed crashes tied to JavaFX initiation orders.
- Writing Projectplan.

Challenges:

- Pinpointing the exact cause of JavaFX launch errors when multiple threads tried accessing GUI elements simultaneously.

What did I learn?

- How to systematically debug a problem.

Future Plans

- implement the rope into the GUI.

Date: April 6/7 2025

Author: William

What I Did Today:

- writing the manual
- writing the project plan with gantt project
- Updating Network Protocol
- Edit README
- implement cs108
- Testing and debugging

Challenges;

- Project Plan and debugging of the main class

Future Plans:

- Unittests
- Rope implementation as an item

Date: April 6 / 7, 2025
Author: Aiysha

What I Did Today:

- **New Game Objects – Key and Door:**
  - Developed a new key object with mechanics similar to the box. The key can be grabbed and thrown, adding to the interactive gameplay.
  - Implemented a door object that acts as a win condition trigger. When the key touches the door, a winning message is sent, marking a successful game completion.
  -
- **Chat Integration:**

    - Integrated the chat features into the main game interface, so players can communicate seamlessly during gameplay.Synchronized chat functionality with other physics-based features, ensuring that both communication and gameplay updates occur in real time.

Challenges I Faced:

- Ensuring smooth integration of the new objects (key and door) into the existing client authority-based synchronisation system.
- Handling the collision detection and interactions between the key and the door to reliably trigger the win condition.
- Merging the new chat functionalities into the main game interface without disrupting the existing physics and synchronisation logic.

What I Learned:

- How to extend existing game mechanics by adding new objects that share similar properties to established ones (like the key with box mechanics).

Future Plans:

- Refinement and Balancing:
    - Further refine the interaction mechanics of the key and door, and ensure that collision detection is reliable under various gameplay scenarios.
    - Balance the new win condition and explore additional gameplay events triggered by object interactions.
- Feature Enhancements:
    - Expand chat features to include more advanced functionalities like private messaging, chat history, and improved UI responsiveness.
- Optimization:
    - Optimize synchronisation and performance of both the physics and chat systems to maintain smooth gameplay in larger multiplayer sessions.
- User Testing:
    - Conduct user testing to gather feedback on the new objects and integrated chat features, then iterate on the design based on the insights gained.

Date: April 6, 2025
Author: Illia

GUI Refinement and Team Merge Session

What did we do today?

Today Sena and I worked on refining the graphical user interface for the Global, Whisper, and Lobby chat tabs. Our focus was on improving both the visual appearance and the usability of these features.

Work Summary:

- We enhanced the design and layout of the chat interfaces to make them more intuitive and visually consistent.
- Functionality was adjusted to ensure the chat areas are responsive and easy to interact with.
- Later in the day, we had a Discord meeting with Aiysha and William, where we all reviewed the changes made individually.
- We tested the new implementations together and ensured compatibility across components.
- After successful testing, we merged the updates into the main branch.
- Finally we have updated all pdf and mark down documents

Challenges We Faced:

- Aligning different GUI components while maintaining consistency across chat modes required several refinements.
- Coordinating changes made by different team members and resolving small merge conflicts took time and careful testing.

What did I learn?

- Regular collaboration and peer review help prevent integration issues.
- A visually clean and functional interface improves both development and user experience.

Date: April 9, 2025

Author: Aiysha

Revamping Throwing and Grabbing with Synchronized Key Presses

What did I do today?

Today marked a major milestone in how our player mechanics work. Initially, Illia reworked the player logic so that key press events are synchronized across clients instead of continuous position updates. This change has greatly improved consistency and reduced desynchronization issues. After Illia's foundational work, he and I collaborated to further rework the player—adjusting the control flows and integrating the new key-press model across our codebase.

Once the player logic was updated, William implemented a robust grabbing feature. His grabbing system reliably identifies when a nearby object or player should be picked up and carried. Building on these improvements, I added the throwing feature. In my version, I introduced a two-key mechanism:

- Pressing **F** toggles throwing mode. When I press F for the first time, it enables throwing mode and a red indicator line oscillates in a windshield-wiper style to visualize the throw angle (between 0° and 180°). If I press F again while already in throwing mode, the mode is canceled.
-
- While in throwing mode, pressing **R** executes the throw. The current throw angle—updated continuously during the oscillation—is used together with a controlled magnitude (reduced to ensure a gentle throw) to compute the throw vector, which is then applied to the grabbed object.

Additionally, I refined the jumping mechanic so that if the player is carrying an object, the jump force is halved to better simulate the added weight.

By shifting to key-press synchronization, every client now processes the same input commands locally, ensuring that our game behaves consistently across the network.

Challenges I Faced

- Transitioning to Key-Press Synchronization:
  Our original system synchronized continuous movement data, which led to severe desynchronizations and unplayable grabbing/throwing behavior. Illia's rework of the player to synchronize key presses instead required a complete overhaul of our control logic. Illia and I had to carefully rework multiple parts of the player code to ensure that all clients would correctly interpret and simulate the same inputs.

- Integrating New Grabbing with Throwing:
  William's grabbing feature was a huge improvement over our older approach.

However, merging his grabbing logic into the reworked player structure demanded significant adjustments. I had to ensure that the grabbing state was maintained reliably so that when it came time to execute a throw, the correct object was affected.

- Managing Input Overload:
  Initially, I tried using the F key for both activating and executing the throw. This overloaded approach resulted in unpredictable behavior—objects were being thrown with excessive force. Splitting the tasks by using the F key to toggle throwing mode and the R key to execute the throw required some rethinking but ultimately led to a much more controlled feature.

- Oscillation Tuning and Force Calibration:
  Fine-tuning the oscillation of the throw angle and calibrating the throw magnitude were both challenging. I experimented with various rates and limits until I found parameters that produced a visually clear and gameplay-friendly indicator, as well as a throwing force that wasn't overpowered.

What did I learn?

Today's process reinforced that sometimes resolving deep-rooted synchronization issues requires a complete rethink of the approach. Moving away from continuous movement updates to a key-press model not only solved our desync issues but also opened a path for more consistent feature implementation.
I learned that overloading a single key for multiple actions can lead to serious, unintended side effects. By splitting the responsibilities—using F for toggling throwing mode and R for executing the throw—the control scheme became more intuitive and predictable.
Furthermore, integrating William's grabbing logic demonstrated the power of modular development. Even though we all worked separately on our pieces, our contributions came together seamlessly to create a much improved gameplay experience.

Future Plans

- Extensive Playtesting and Refinement:
  I plan to conduct further testing under various network conditions to optimize responsiveness and ensure that key-press synchronization is rock-solid. This will help in fine-tuning the grabbing and throwing mechanics to address any remaining latency or responsiveness issues.

- Control Smoothing:
  Even with the new two-key system for throwing, the controls still feel somewhat clunky. I intend to adjust the oscillation mechanics—refining the angle range, oscillation rate, and overall input mapping—for a smoother experience.

- Performance Optimization:
  Although the change to key-press synchronization reduced lag, some minor latency remains. Future efforts will focus on enhancing our network code and client prediction to further minimize any lag.

- Enhanced Visual and Audio Feedback:
  I want to introduce richer visual cues—such as particle effects, additional UI indicators, or more detailed throw animations—and corresponding sound effects when entering throwing mode, canceling it, or executing a throw. This will improve both the clarity and satisfaction of these interactions.

- Interface-Driven Object Features:
  Another exciting future plan is to adopt a more modular approach by having game objects implement standard interfaces (like IGrabbable, IThrowable, etc.). With this strategy, any object that implements these interfaces will automatically inherit grabbing and throwing capabilities without extra hardcoded logic. This will make our codebase more modular and extensible, allowing us to introduce new interactable objects more easily.

- Further Feature Integration:
  I'm considering additional interactions between grabbing, throwing, and player attributes. For example, different objects might affect a player's movement or abilities when carried, further enriching the gameplay experience.

Date: April 16 2025

Author: William

Fixing Mainclass

What did I do?

I fixed the mainclass that you only need to type the port for the server.

What did I learn?

I learned that I can use the port 0.0.0.0 for it to work.

Future plans:

implementing Win condition

Date: April 14–17, 2025
Author: Sena

Gui Improvements and Integration

What did I do during this period?

Between **April 14 and April 17**, I focused on improving the graphical user interface (GUI) and preparing the project for smoother user interaction and better layout management.

Work Summary:

- Replaced the old layout structure with a VBox-based `mainUIPane`, ensuring a more flexible and organized design.
- Implemented automatic adjustment of the platform size according to the main pane dimensions for better responsiveness.
- Added a new pane for starting the game, including buttons for character selection.
- Moved the game ID field, text field, and create object button to the administration panel for cleaner separation of functionality.
- Adjusted the layout of all chat panels to create a more consistent and user-friendly appearance.
- Initially set the Start Game button to be visible; character selection is now revealed only after the game has started.
- Added a duplicate name check to prevent users from creating multiple game objects with the same name.
- Successfully merged the GUI-configuration branch into the main branch without any conflicts, following synchronization adjustments.

Challenges I Faced:

- Ensuring dynamic resizing of the platform area without breaking the overall GUI layout required several testing iterations.
- Keeping a smooth and logical flow between game creation, character selection, and starting the game demanded careful layout planning.
- Verifying consistency after merging and avoiding GUI regressions was a meticulous process.

What did I learn?

- Using flexible layout structures like **VBox** simplifies the management of dynamic UI elements.
- Early validation, such as checking for duplicate names, improves the robustness of user interactions.
- Careful branch synchronization and clean merging practices save significant time during integration.

Future Plans:

- Conduct full GUI testing with server communication enabled.

- Prepare the system for internal demo sessions and feedback gathering.

Date: April 23, 2025
Author: Sena

Implementing Unit Test for Chat Command Handlers

What did I do today?

I focused on strengthening the reliability of the chat-related components in our project by developing comprehensive unit tests.

Work Summary:

- Implemented unit tests for the WhisperCommandHandler class to verify that private message functionality behaves as expected.

- Created unit tests for the ChatLobbyCommandHandler class to ensure proper handling of lobby-specific chat messages.

- Developed unit tests for the ChatGlbCommandHandler class to test the correct broadcasting of global chat messages to all players.

Challenges I Faced:

- Designing meaningful test cases that cover both normal and edge cases in server-client communication required careful attention.

- Mocking dependencies and setting up isolated testing environments for command handlers took several iterations to fine-tune.

What did I learn?

- Writing thorough unit tests in Java significantly improves code stability and simplifies future refactoring.

- Clear responsibility separation between handlers makes both development and testing more efficient.

Future Plans:

- Continue increasing unit test coverage for other command handlers.
- Start integrating unit tests into an automated build process

- Prepare for upcoming integration testing that will involve end-to-end communication flows.

Date: April 26, 2025
Author: Aiysha

Implementing Unit Tests for Game, Client, Server, GameObject

What did I do today:

- Write unit tests for our core game components:
- GameContext: Verified singleton behaviour, session lookup, adding/retrieving games
- Client: Tested server-address configuration, message-sending (static vs best effort), ACK delegation, local state updates (updateLocalClientState)
- Server: Covered singleton instance, makeResponse(), unique-name generation, enqueuing outgoing messages, game-session synchronization, and reset-handling fallback for unknown commands.
- Game: Ensured ID/name initialization, user-management, startup logic, tick counting, and routing of incoming commands.

Challenges I Faced:

- Mocking and StubbingCreating realistic test doubles for networking components (e.g., DatagramSocket, AckProcessor) to verify packet contents without launching real sockets.
- Mocking Networking Internals: Verifying DatagramPacket contents and AckProcessor interactions without spinning up real sockets.
- Private Queue Inspection: Safely resetting and inspecting private LinkedBlockingqueue fields in Server and others via test-only reflection utilities.
- Taming Background Executions: Shutting down AsyncManager between tests to avoid RejectedExecutionException and ensure a clean executor lifecycle.

What I Learned:

- Selective Reflection
  Confine reflection to narrow helper methods in tests rather than sprinkling setAccessible(true) everywhere.
- Executor Control
  Always pair start with a test-driven shutdown to avoid leaks and flakiness in concurrency-driven code.

Future Plans:

- Physics and Collision Tests: Unit-Test Gravity and friction, Collision scenarios among GameObject subclasses once their stubs are in place.
- Deterministic Timing: Refactor AyncManager to accept a testable scheduler or clock abstraction to eliminate real-time sleeps in tests.

Date: April 27, 2025
Author: Aiysha

Implementing Unit Tests for Physics and Collision, GameObject

What did I do today:

- Added PhysicsAndCollisionTest: This covers the GravityEngine's behaviour by verifying it applies gravity only to IGravityAffected objects and ignores plain GameObjects.
- Exercised collision resolution logic in GameObject.resolveCollision(...) with three scenarios:
    1. Symmetric collision between two movable objects.
    2. Both static, ensuring no movement
    3. Vertical overlap, confirming Y-axis resolution.
- Defined minimal stub classes (StubGravityObject, StubPlainObject, StubCollisionObject) to isolate and these algorithms without dragging in full game logic.

- Write GameObjectTest using a StubGameObject subclass to exercise all concrete logic in GameObject. Covered:
    1. Positioning (setPos, getPos)
    2. Grab/release state
    3. Selected flag getters/setters
    4. Collidable and movable flags
    5. Incomming-message queue size limit and processing
    6. Command queue execution
    7. Intersection tests for overlapping / non-overlapping bounding boxes
    8. Collision resolution in both symmetric and one-static scenarios

9. extractGameId helper behavior.

Challenges I Faced:

- Abstract base testing: I needed a concrete stub class that implements all abstract methods, but does nothing except record calls.
- Internal queues: couldn't directly inspect the private incoming-message queue, so I drove it via processIncomingMessages() to verify overflow behaviour.

What I Learned:

- How to Stub classes: Stubbing abstract classes is invaluable for isolating and validating shared logic before writing any real subclasses.
- Collision math must be tested in both directions (symmetric vs one-static) to catch off-by-half-overlap errors.

Future Plans:

- Write Unit Test that Starts a server, client and runs a game.

Date: April 27, 2025

Author: Sena

Expanding Unit Test Coverage for Command Handlers

What did I do today?

I focused on expanding the unit test coverage for our server-side command handlers in the Java project, using Mockito for mocking dependencies.

Work Summary:

- Implemented unit tests for critical server command handlers such as LoginCommandHandler, LogoutCommandHandler, CreateCommandHandler, JoinGameCommandHandler, and StartGameCommandHandler.
- Added unit tests for communication-related handlers like WhisperCommandHandler and ChatLobbyCommandHandler.
- Verified the correct broadcasting of messages in CreateGameCommandHandlerTest, ensuring messages are sent with the correct content and recipients.

- Used Mockito to mock server-side dependencies and simulate client-server interactions accurately.
- Strengthened validation for session management, object creation, and chat message handling.

Challenges I Faced:

- Setting up detailed mock environments with Mockito to simulate different server behaviors was complex and required careful scenario planning.

What did I learn?

- Utilizing Mockito effectively makes it easier to isolate units for precise testing in Java projects.
- Comprehensive unit testing enhances the system's reliability and greatly simplifies future debugging and feature additions.

Date: April 27, 2025

Author: Aiysha

Implement game starting Unit Test and expand other Unit Tests

What did I do today?

I wrote the StartGameTest unit Test for the headless client (which is a case I've also added to the ThinkOutsideTheRoom main class). I also edited the CreateGoCommandHandlerTest to stub missing maps and lists. I Upgraded the DeleteGoCommandHandlerTest for correct stubbing. And I refined BaseTest to simplify AsyncManager mocking.

Work Summary

1. **StartGameTest**

- Created a StartGameTest that spins up the real server on UDP port 8888.
- Added a "headless" client mode (client-headless) to the main entry point so the test can drive the client without launching JavaFX.

- In the test, launched the headless client with arguments (host, port, username, gameName), read its stdout, and asserted it prints Joined game: MyGame within a 5 s timeout.

2. **CreateGoCommandHandlerTest Enhancements**

- Stubbed out server.getClientsMap(), fakeGame.getGameObjects(), and gsm.getAllGameSessions() to return empty—but non-null—collections, preventing NullPointerExceptions in the handler logic.

- Enabled @MockitoSettings(strictness = LENIENT) so that necessary stubs for the "too few parameters" scenario aren't flagged as "unnecessary."

- Added a Captor<Message> to verify that the broadcasted response carries the correct session ID, object type, and name.

DeleteGoCommandHandlerTest Adjustments

- Ensured the mock Server returns a non-null GameSessionManager and that its getClientsMap() returns a valid map.

- Verified that when deleting an object by ID, the handler both removes it from the game's gameObjects list and broadcasts the deletion message exactly once.

- **BaseTest Simplification for AsyncManager**

- Removed overly complex static mocking of AsyncManager.runLoop and run(Callable).

- Kept only a basic stub for AsyncManager.run(Runnable) in @BeforeAll, so that any asynchronous tasks submitted during handler tests execute immediately on the calling thread.

- This keeps tests fast and avoids RejectedExecutionExceptions without touching production code.

Challenges I Faced:

- NullPointerExceptions in Handler Tests
  Stubbing out every null-returning collection (getClientsMap(), getGameObjects(), getAllGameSessions()) to keep the handler logic from blowing up.

- Mockito Strictness
  Balancing between strict stubbing (to catch unused mocks) and lenient mode (for tests that only exercise part of the stubs, e.g. “too few parameters” cases).

- Async vs. Sync in Tests
  Ensuring asynchronous code (via AsyncManager) runs deterministically on the calling thread, without introducing flaky test timing or executor shutdown errors.

- Headless vs. GUI Mode
  Refactoring main(...) to support a “headless” client that can be driven from a JUnit test, without pulling in JavaFX or hanging on Application.launch().

- Process I/O and Timing
  Reading stdout from the spawned client process reliably, adding the right amount of sleep—and avoiding brittle timeouts—so the integration test neither false-fails nor hangs indefinitely.

- Duplicate Initialization
  Diagnosing why Alfred and Gerald appeared twice, discovering the double call to initializeLevel(...), and then introducing a guard flag to prevent repeated setup.

- Variable Scope in switch
  Running into “variable already defined” errors when adding the headless mode, solved by wrapping each case in its own {…} block.

- ShadowJar Plugin Quirks
  Wrestling with Gradle’s plugin ordering and task configuration so that shadowJar actually ran and produced the expected -all.jar output.

What I Learned:

- I learned, that sometimes just checking the order or amount of things is the solution, instead of turning the whole project upside down. (There was a bug with double game objects when starting a Level. The problem was, objects where initialized twice).
- Always stub out every collection or map that a handler might iterate over - otherwise you’ll hit NullPointerExceptions in unit tests.
- Ripping each case in its own block prevents duplicate-variable errors and keeps code clear.

Date: 22. - 27. April

Author: William

Implementing Highscore

What did I do?

I implemented a winning condition, which you achieve by touching the key to the door. After you completed the level a txt. file will be created with all the users in the lobby and the time, which it took to complete the level.

Work summary:

- Implementing Win condition
- Implementing Timer class
- implementing highscore
- A lot of debugging

What did I learn?

I learned to check the order and use for every function, because there was a bug, which the key couldn’t be thrown. I also learned to use flags and System.out.println more often for debugging.