CS 555 Project: Quantum Knapsack

by

Enoch Chan, Benedict Martinez, Benjamin Moks, Katie Ng, Kyle Ponte, Kent Quach

Stevens Institute of Technology

March 11, 2025

CS 555 Project: Quantum Knapsack

Enoch Chan, Benedict Martinez, Benjamin Moks, Katie Ng, Kyle Ponte, Kent Quach
Stevens Institute of Technology

The following table (Table 1) should be updated by authors whenever major changes are made to the architecture design or new components are added. Add updates to the top of the table. Most recent changes to the document should be seen first and the oldest last.

Table 1: Document Update History

Date	**Updates**
01/22/2025	Student A: • Updated sswManual.tex with *newcommand(s)*{} for easier references of requirements, figures, and other labels.
01/29/2025	Student A and C: • Added chapters on requirements (Chapter 4) and user stories (Chapter 8).
02/25/2025	Student B and C: • Added chapter on development plan (Chapter 3) and glossary.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Team Declaration

Chan, Martinez, Moks, Ng, Ponte, Quach

Quantum Knapsack presents a twist to the famous knapsack optimization problem by introducing basic quantum concepts in a gamified manner. Given a knapsack (with known capacity) and an array of qubits (of varying weights and values), the player's goal is to select the optimal set of qubits that will maximize their value without exceeding the knapsack's capacity.

In this challenge, the value of each qubit remains undetermined until ”measured” (triggered by adding the given qubit to the knapsack). This simulates **superposition**, a phenomenon describing the qubit's ability to exist in multiple states simultaneously. Eventually, the player is exposed to random **entanglement**, where connected qubits will resolve to either one of their expected values. In later rounds, the knapsack will also be at risk of **decoherence**, which decreases its capacity. The player is encouraged to think strategically about the trade-offs, as well as how to prepare and adjust to these uncertainties. Once they are satisfied with their set of items, they can collect the total value of their knapsack to use as resources for the next round. Players will be scored on their strategy based on the ”stability” of their knapsack.

The game is split into four chapters to ensure a steady progression through basic knapsack mechanics, superposition, entanglement, and finally decoherence. Every chapter will include explanations to teach the topic. As the player progresses, the resource they collect can be spent on enhancements. If bandwidth allows, we also hope to incorporate creative variations (i.e. knapsack types) so players can try different challenges.

1.1 Mission Statement

Mission Statement: To address the learning curve associated with quantum computing, Quantum Knapsack introduces foundational concepts using a variation of the classic knapsack problem. Through steady progression to higher-level concepts, our mission is to design puzzles that spark interest in quantum computing without advanced math prerequisites.

Key Drivers: Our success factors include:

1. **Portrayal of Quantum Concepts**: Representing select quantum mechanics (superposition, entanglement, decoherence) clearly and accurately through gameplay elements

2. **Digestibility**: Designing a narrative to teach concepts in increasing complexity
3. **Engagement**: Processing both client and end-user feedback across several iterations to ensure that game is educational and engaging
4. **Scalability**: Structuring the game to be open-ended and adaptable, with potential to add new puzzles/challenges

Key Constraints: The most apparent constraint is our inability to perform true quantum computing due to limited quantum resources. Hence, these concepts must be simplified to fit within a typical server's computing power, while still being representative of their theoretical mechanics.

Milestones:

1. **Product Conceptualization**: Finalized game development document and technical plan
2. **Core Development**: Functional knapsack system and superposition mechanics
3. **Concept Expansion**: Integrated mechanics for entanglement and decoherence
4. **Visual Design**: Completed user interface
5. **Final Testing**: Rigorous testing for final bug fixes and refinement
6. **Launch**: Full deployment of product to be available for early adopters

1.2 About the Team

1.2.1 Enoch Chan

Enoch is pursuing a Bachelor of Engineering in Software Engineering. He is particularly fascinated by how intelligent systems can enhance automation, decision-making, and user experience in various domains. His interests extend to machine learning, algorithm optimization, and data-driven applications, where he enjoys exploring innovative ways to improve system efficiency. This project presents an exciting opportunity for Enoch to apply his knowledge, grow as a developer, and contribute meaningfully to the team's success.

1.2.2 Benedict Martinez

Benedict is pursuing a Bachelor of Engineering in Software Engineering with a minor in Computer Science. He is driven by a deep curiosity to understand how things work. He yearns to continue growing his skills in software engineering, engineering management, and collaborating with like-minded individuals to bring his ideas to life. Benedict values collaboration and continuous learning, making him eager to apply agile methodologies and contribute meaningfully to this project.

1.2.3 Benjamin Moks

Ben is pursuing a Bachelor of Engineering in Software Engineering, with a keen interest in system architecture, scalable software design, and real-time computing. He enjoys tackling complex problem-solving challenges, optimizing software performance, and ensuring seamless integration between different technologies. This project provides an exciting opportunity for him to refine his technical skills, collaborate effectively, and contribute to building a robust and user-friendly solution.

1.2.4 Katie Ng

Katie is pursuing a Bachelor of Science in Computer Science with a minor in Quantitative Finance. She is interested in financial applications of software engineering, specifically focused on optimization problems in equity trading. Throughout her academic career, she has worked to polish her technical skill set, with her greatest strengths being web development and database management. Having had some exposure to scrum in her previous internships, she is excited to learn and practice agile methodologies more formally!

1.2.5 Kyle Ponte

Kyle is currently pursuing a Bachelor of Engineering in Software Engineering, with a growing interest in data science and machine learning. He thrives on tackling complex challenges and uncovering innovative solutions, and he believes that collaboration and diverse perspectives drive meaningful progress. Eager to continually refine his skill set, Kyle sees this project as an opportunity to apply his knowledge, practice agile methodologies, and assist in the team's success.

1.2.6 Kent Quach

Kent is currently pursuing a Bachelor of Science in Computer Science with a growing interest in full-stack web development. He thrives in collaborative environments, working alongside others who share the same vision of creating intuitive and impactful digital experiences. As a soon-to-be graduate, he is eager to keep learning, refining his skills, and contributing to meaningful projects. This project presents an opportunity for Kent to apply himself and further his development.

Chapter 2

Weekly Reports

Every week, the team is expected to add a new report.

2.1 Week Report 1 (02/18/2025)

What We Did This Past Week

This past week, we initiated product conceptualization to develop an overview of our Quantum Knapsack game. We discussed several viable options for integrating these concepts into our game. For example, entanglement could have been represented as a ”buy one, get one” event in which users must add entangled qubits as a package. Ultimately, our decisions were guided by how accurately it represented quantum concepts, and in moments of uncertainty, we relied heavily on our understanding of our target audience (i.e. what would be most intuitive to them). Our deliverables include our Introduction (*Chapter 1*) and Development Plan (*Chapter 3*), which we will continue to revise with our client.

What We Will Do Next Week

This upcoming week, we hope to review our product idea with our client to refine our success criteria. Based on these discussions, we will further specify requirements before moving onto task delegation on Jira.

2.2 Week Report 2 (02/25/2025)

What We Did This Past Week

This past week, our team confirmed the project description with the client via Slack. We then created a detailed project description and populated our Jira backlog with well-defined user stories, prioritizing them based on team discussions and assigning story point estimates. A subset of user stories were selected for Sprint 1, which will start on February 27th for a two-week duration, ending on March 13th. The planning, architecture, use cases, and user story sections were all updated accordingly, as well as some edits to our requirements. Furthermore, we converted two of our completed user stories into user cases (refer to Chapter 6) in our documentation. For these two examples, use cases were more effective because they involved interactions with system states and required defining system constraints, such as error handling when weight limits are exceeded. Finally, a Planning Poker was conducted in an effort to determine time estimates for each user story.

During our Planning Poker session, the team estimated the effort required for our user stories, and there were no major disagreements. Everyone had a clear understanding of the scope and complexity of the tasks, allowing us to reach a consensus quickly. However, if disagreements were to arise in the future, we would address them by discussing and justifying our estimates, ensuring that each team member has the opportunity to explain their reasoning and highlight any complexities they foresee. If necessary, we would revisit the user story to clarify requirements and align on expectations. In cases where differences persist, we could opt for an average estimate or conduct a re-vote after further discussion. Additionally, if the team requires further insights, we would seek input from a senior developer or someone with domain expertise. By maintaining open communication and using these strategies, we can ensure more accurate and agreed-upon estimations in future sessions.

What We Will Do Next Week
This upcoming week, we will begin implementation of Sprint 1 by initiating development on the prioritized user stories and their associated subtasks. We plan to monitor progress and quickly address any blockers that may arise. Additionally, we will review the feedback provided by the client on our user stories, architecture, and planning documents, and integrate any necessary adjustments to ensure alignment with client expectations.

2.3 Week Report 3 (03/04/2025)

What We Did This Past Week
This past week, our team has started implementing Sprint 1 by setting up the website and GitHub repository for the game. We have also started development of the knapsack functionality, which is a core feature of the game. This includes defining the qubit object, implementing selection/deselection, implementing updates to the knapsack, implementing superposition of a qubit, and error handling. Testing was also implemented for the knapsack functionality using Vitest, a JS testing framework.

What We Will Do Next Week
This upcoming week, we will conclude our implementation of Sprint 1 by completing the following tasks:

- Finish setting up the MERN project
- Creating a landing page for the application
- Adding a button to finalize knapsack value

2.4 Week Report 4 (03/11/2025)

What We Did This Past Week

This past week, our team has been working to finish up Sprint 1 by finishing the MERN project setup, creating the landing page for the application, and adding a button which will finalize the knapsack value for the player.

Screenshot of Jira Backlog

Screenshot of Burndown Chart

What We Will Do Next Week

This upcoming week, we will be moving into Spring Break. Sprint 2 will begin the following week on Thursday, March 20th.

Chapter 3

Development Plan

– Chan, Martinez, Moks, Ng, Ponte, Quach

3.1 Introduction

The Development Plan outlines the methodology, technical specifications, and milestones for building Quantum Knapsack. Our Requirements Document (*Chapter 4*) and System Architecture Review (*Chapter 5*) are living documents that provide a structured approach to delivery.

Our problem statement stems from acknowledging the high barrier to entry in quantum. Hence, our primary success criteria includes accurate portrayal of quantum concepts (decided by the client), as well as engagement and educational impact (collected from student feedback).

Quantum Knapsack will be a web-based game developed on the MERN stack. Technical and non-technical requirements will be defined during the conceptualization stage and will serve as a guide for development, though specifics are adaptable to changing needs. A functional prototype is expected after the development stages, after which we will move onto testing and launch. As with learning, implementation of complex concepts is also dependent on basic building blocks, so our development phase will mirror our four-part gameplay progression outlined in Chapter 1.

3.2 Roles and Responsibilities

These vary for each type of product. For small projects, folks may serve multiple roles. This is a list of common roles we have used for software development:

1. Development Lead: Benjamin Moks
2. Architect: Katie Ng
3. Developers: Enoch Chan, Benedict Martinez, Benjamin Moks, Katie Ng, Kyle Ponte, Kent Quach
4. Test Lead: Benedict Martinez
5. Tester(s): Kent Quach, Kyle Ponte, Enoch Chan

6. Documentation: Enoch Chan, Benedict Martinez, Benjamin Moks, Katie Ng, Kyle Ponte, Kent Quach
7. Documentation Editor: Kent Quach
8. Designer: Benjamin Moks
9. User Advocate: Enoch Chan
10. Risk Management: Kyle Ponte
11. System Administrator: Kyle Ponte
12. Modification Request Board: Enoch Chan (leader), Katie Ng, Benjamin Moks
13. Requirements Resource: Benedict Martinez
14. Customer Representative: Enoch Chan, Kyle Ponte

3.3 Method

3.3.1 Software

1. Language(s): Javascript, HTML5, CSS3
2. Tools/Environments: Node.js v22.14.0, MongoDB v8.0
3. Frameworks: Express v4.21.2, React v18.0.0
4. Software packages/libraries: Material UI v7

3.3.2 Review Process

1. (Architecture) Our team will validate the technical architecture to guarantee expected performance on classic (non-quantum) computers. Once complete, we will request feedback from our client regarding the proposed algorithms for quantum state representation and randomness.
2. (Security) *TBD, depending on the data collected from users.* If login is not required, we will test against common website vulnerabilities (XSS, SQL injections, CSRF). If login is required, we must also consider how to store user data securely.
3. (Usability) Usability review should come from beta users that will evaluate the product's interface and experience based on a rubric.
4. **Formal review** will be implemented as regular tasks in each epic. Our Test Lead and Testers will be responsible for testing our product against functional bugs and pitfalls, while other Developers will focus on reviewing for code quality.

3.3.3 Build Plan

1. Revision Control System: Git
2. Repository: Github (link)
3. Regularity of the builds: Every 3 days
4. Deadlines for the builds: Night before build, 11:59 PM

3.4 COMMUNICATION PLAN

3.4.1 Heartbeat Meetings

Purpose: To align the full team on the project's overall progress and to facilitate next steps.
Attendees:

- Development Lead and Developers
- Designer
- Scrum Master

Frequency: Weekly, 20 minutes (Exact time TBD)
Agenda:

- Team Updates: Each team member provides a 2 minute update on their assigned tasks.
- Upcoming Tasks: Delegate tasks for the upcoming week.
- Blockers: Discuss any challenges or blockers that may impact progress.

3.4.2 Status Meetings

Purpose: To review progress in the context of larger milestones, and to share updates with the client. If any concerns arise, the team can also use this time to adjust and add requirements.
Attendees:

- Development Lead
- Architect
- Customer Representative
- (Optional) Client

Frequency: Monthly, 40 minutes (Exact time TBD)
Agenda:

- Progress Summary: Share updates regarding completed, in-progress, and scheduled tasks.
- Milestone Alignment: Check status and ensure clear ownership for upcoming deliverables. Adjust priorities if necessary.
- Dependencies: Discuss dependencies that may hinder any individual contributor's progress, and agree upon workarounds or smaller deadlines for relevant team members.
- Future Outlook: Confirm next steps.

3.4.3 Issues Meetings

Purpose: To escalate (and hopefully resolve) critical issues. Can be initiated by any team member that discovers a problem requiring urgent resolution.
Frequency: As needed
Agenda:

- Issue Overview: Define the issue, especially its cause and impact.
- Resolution Plan: Develop a course of action to resolve the issue, identify affected tasks, and adjust accordingly.
- (Optional, if time allows) Testing: Validate the efficiency of the agreed-upon resolution plan.

3.5 Timeline and Milestones

3.5.1 Product Conceptualization

Deliverables: A comprehensive Game Development Document (GDD) explaining the game's design and mechanics, as well as a Technical Plan detailing the software architecture.
Tasks:

- Define representation quantum mechanics (superposition, entanglement, decoherence) through gameplay elements.
- Agree upon each team member's experience, individual goals, and project expectations.
- Approval of game design and technical architecture by client.

Begin Time: Week 1
End Time: Week 2

3.5.2 Core Development

Deliverables: A functional gameplay loop comprised of the general knapsack system and superposition mechanics.
Tasks:

- Design a site map, and set up the boilerplate application.
- Develop the knapsack inventory system.
- Implement the superposition mechanic (unresolved qubit values).
- Ensure mathematical accuracy of implemented algorithms (proper randomization, correct optimization, etc).

Begin Time: Week 2
End Time: Week 5

3.5.3 Concept Expansion

Deliverables: Integrated entanglement and decoherence mechanics in gameplay.
Tasks:

- Implement the entanglement mechanic (resolving linked qubits to one value).
- Implement the decoherence mechanic (random events to decrease knapsack capacity).
- Again, ensure mathematical accuracy of implemented algorithms.

Begin Time: Week 5
End Time: Week 7

3.5.4 Visual Design

Deliverables: A polished user interface allowing end-users to play.
Tasks:

- Evaluate current representations of abstract concepts (superposition, entanglement, decoherence) and re-design if necessary.
- Design wireframes in Figma to visualize different screens in gameplay.
- Perform usability testing.

Begin Time: Week 7
End Time: Week 8

3.5.5 Final Testing

Deliverables: A rigorously tested product that is ready for deployment to beta users.
Tasks:

- Compile a comprehensive document detailing all relevant tests.
- Test all possible user paths, including both expected and unexpected behaviors.
- Resolve identified bugs.

Begin Time: Week 8
End Time: Week 9

3.5.6 Launch

Deliverables: Fully deployed product!
Tasks:

- Deploy the game to a web server that should be easily accessible to selected users.
- Monitor product performance, which is now handling new users.

Begin Time: Week 9
End Time: Week 9

3.6 Testing Policy and Plan

Our testing approach follows an iterative process, ensuring that each development stage is validated before moving forward. The stopping criteria for each testing phase will be when the system meets our quality benchmarks, detailed here.

3.6.1 Unit Testing

Start: During core development (Week 2)
Purpose: Validate individual functions and components to ensure correctness and reliability.
Stopping Criteria: When all key functions pass predefined unit test cases with a 95% pass rate.

3.6.2 Integration Testing

Start: After major components are developed (Week 5)
Purpose: Ensure smooth interaction between different modules and prevent integration issues.
Stopping Criteria: When all integrated components function correctly with minimal defects.

3.6.3 System Testing

Start: Once the full system is functional (Week 7)
Purpose: Verify that the complete system meets the specified requirements and handles expected user interactions.
Stopping Criteria: When the system operates as expected under normal and stress conditions, including bad input.

3.6.4 Acceptance Testing

Start: Before final deployment (Week 8)
Purpose: Ensure the system aligns with client expectations and functional requirements.
Stopping Criteria: When the client formally approves the product for release.

3.6.5 Regression Testing

Start: After each major update or bug fix
Purpose: Confirm that new changes do not introduce unexpected issues.
Stopping Criteria: When all previously working features remain functional after updates. (Regression testing will be reintroduced with every update as long as the product continues to be developed.)

3.6.6 Security Testing

Start: During system testing (Week 7)
Purpose: Identify and mitigate security vulnerabilities, such as unauthorized access.
Stopping Criteria: When all critical security threats are resolved.

3.7 Risks

Our risk management approach involves **early identification, continuous monitoring, and structured mitigation plans**. Risks will be tracked using Jira, with each risk assigned a severity level

(Low, Medium, High) and reviewed in our weekly meetings.

3.7.1 Complexity of Quantum Mechanics

High. Quantum mechanics concepts may be misrepresented due to their complexity. To mitigate this, we will regularly consult with our client and domain experts to ensure accurate representation.
Tracking: Check-ins with the client and close documentation of changes in Jira.

3.7.2 Team Availability and Workload Balancing

Medium. Scheduling conflicts may affect the team's progress. To address this, we will assign backup roles for critical tasks and follow agile sprint planning to ensure fair workload distribution.
Tracking: Weekly stand-up meetings to assess progress and reassign tasks as needed.

3.7.3 Security Vulnerabilities

High. All web applications are vulnerable to common security threats. To prevent this, we will follow best security practices such as input validation and XSS defense.
Tracking: Security reviews and penetration testing will be conducted before each release.

3.7.4 Browser Compatibility

Medium. The game may not function consistently across different browsers and devices. To ensure compatibility, we will conduct cross-browser testing (Chrome, Firefox, Edge, Safari).
Tracking: Manual testing before deployment (and perhaps automate testing if possible).

3.8 Assumptions

- Team Skill Set: Assume that our team collectively has the necessary skills to build this game. If conflicts arise, we may need to adjust our software architecture.
- Platform Compatibility: Assume that our computational requirements can be handled by the user's server.
- Player Learning Curve: Given that our intended audience is K-12, assume that players have minimal prior knowledge of quantum computing. The game is designed to be simple to provide a gentle learning curve.
- UAT Participation: Assume that sufficient participants are available and willing to provide feedback on our product, and that this feedback provides actionable insights for our team.

3.9 DISTRIBUTION LIST

3.9.1 Recipients

- Development Team: Enoch Chan, Benedict Martinez, Benjamin Moks, Katie Ng, Kyle Ponte, Kent Quach
- Instructor and TA: Prof. Yu
- Client: Prof. Chen, Amiratabak Bahengam, Iser Pena

Chapter 4

Requirements

– Chan, Martinez, Moks, Ng, Ponte, Quach

4.1 Stakeholders

4.1.1 Customers

The primary customers of Quantum Knapsack are students, educators, and quantum computing enthusiasts who want to learn fundamental quantum mechanics concepts in an interactive and engaging way. The game is designed for individuals with minimal prior knowledge of quantum computing, making it accessible to K-12 students, college students, and self-learners. Educators can also use the game as a teaching tool to introduce quantum computing concepts in a gamified format.

4.1.2 Sponsors

At this time, Quantum Knapsack does not have any official sponsors. However, the project is developed with guidance from faculty at Stevens Institute of Technology, ensuring alignment with educational and technical best practices. Future sponsorship opportunities may be explored to support further development and scalability.

4.1.3 Competitors

Quantum Knapsack competes with other educational games and platforms that aim to teach quantum computing concepts. Notable competitors include:

- **IBM Quantum Experience** – Provides interactive quantum programming tutorials but lacks gamification
- **Quantum Chess** – A game that integrates quantum mechanics into a strategic board game but focuses more on quantum physics rather than optimization problems
- **Qiskit Textbook Simulators** – Offers direct exposure to quantum programming but requires prior knowledge of Python and Qiskit
- **Quantum Game with Photons** – A puzzle-based game that teaches quantum mechanics concepts like superposition and entanglement.

Unlike these competitors, Quantum Knapsack differentiates itself by integrating quantum mechanics with a well-known optimization problem (the knapsack problem) in a gamified learning environment, making abstract quantum principles more accessible and intuitive.

4.2 Key Concepts

4.2.1 Qubit Superposition Requirement

A placeholder description for the qubit superposition requirement.

The following terms will be used consistently throughout this document and the project:

- **Quantum Knapsack Problem**: A spin on the knapsack optimization problem that incorporates quantum computing concepts such as superposition, entanglement, and decoherence.
- **Qubit**: The basic unit of quantum information. In the game, qubits can represent more complex states by having varying weights and values that influence the player's strategy.
- **Superposition**: A principle of quantum mechanics where a qubit can exist in multiple states simultaneously. In the game, this is represented by qubits having uncertain values.
- **Entanglement**: A quantum phenomenon where the state of one qubit is dependent on the state of another. In the game, this is simulated by linking qubits whose values resolve together.
- **Decoherence**: The process by which a quantum system loses its quantum properties. In the game, decoherence introduces random events that decrease the knapsack's capacity.
- **Knapsack Capacity**: The maximum weight that the knapsack can hold.
- **Measurement**: In quantum computing, measurement forces a qubit to collapse to a definite state. In the game, this occurs when a qubit is added to the knapsack (or after a certain time).
- **Stability Score**: A metric used to evaluate the player's strategy. It is based on how well the player manages the quantum effects (superposition, entanglement, decoherence) to maintain a stable and valuable knapsack.
- **Iterative Learning**: The game is structured in chapters that progressively introduce new concepts, allowing players to build their understanding of quantum mechanics slowly.

4.3 User Requirements

User requirements define what the Quantum Knapsack system should provide from the user's perspective. These are written in natural language and may be supported by informal diagrams such as use case diagrams or user stories.

Each requirement is assigned a priority level:

- **Must-Have**: Essential for the system's core functionality.
- **Should-Have**: Important but not mandatory for initial deployment.
- **Could-Have**: Desirable features to be implemented if time and resources permit.
- **Would-Have**: Optional features for potential future development.

Table 4.1: User Requirements

Requirement ID	**Requirement Description**	**Priority**	**Linked Use Case(s)**
REQ-U1	The system shall allow users to register and log in to track their progress.	Could-Have	. . .
REQ-U2	The system shall provide an interactive tutorial explaining quantum computing concepts.	Must-Have	. . .
REQ-U3	Users shall be able to select/deselect qubits to add/remove to their knapsack.	Must-Have	. . .
REQ-U4	The system shall display the properties of each qubit (e.g., weight, value, probability).	Must-Have	. . .
REQ-U5	The system shall introduce randomness in qubit values upon selection to simulate superposition.	Must-Have	. . .
REQ-U6	The system shall implement an entanglement mechanic where linked qubits collapse together.	Should-Have	. . .
REQ-U7	The system shall decrease knapsack capacity due to random decoherence.	Should-Have	. . .
REQ-U8	Users shall be able to view their past game sessions and performance analytics.	Could-Have	. . .
REQ-U9	The system shall allow users to compete with others on a leaderboard.	Could-Have	. . .
REQ-U10	The system shall allow customization of game difficulty levels.	Would-Have	. . .

4.4 System Requirements

System requirements define the technical and functional constraints that the Quantum Knapsack system must meet to satisfy user needs. These requirements specify system behaviors, interactions, and constraints to ensure proper functionality.

Requirement ID	**Linked UC(s)**	**Name and Description**
REQ-S1	UC1	The system shall allow users to register and log in to track their progress.
REQ-S2	UC2	The system shall provide an interactive tutorial explaining quantum computing concepts before gameplay begins.
REQ-S3	UC3	The system shall enable users to select/deselect qubits to add or remove from their knapsack.

REQ-S4	UC4	The system shall display the properties of each qubit (e.g., weight, value, probability) before selection.
REQ-S5	UC5	The system shall introduce randomness in qubit values upon selection to simulate quantum superposition.
REQ-S6	UC6	The system shall implement an entanglement mechanic where linked qubits collapse together when measured.
REQ-S7	UC7	The system shall decrease knapsack capacity due to random decoherence events.
REQ-S8	UC8	The system shall store and display past game sessions and performance analytics.
REQ-S9	UC9	The system shall allow users to compete on a leaderboard displaying high scores.
REQ-S10	UC10	The system shall allow users to adjust game difficulty levels and customize gameplay settings.

4.5 Non-functional (Quality) Requirements

The Quantum Knapsack system must meet several non-functional requirements to ensure performance, usability, and scalability. These requirements define the system's architecture and operational constraints.

4.5.1 Performance Requirements

- **Response Time:** The game should respond to user actions (for example, adding or removing a qubit) within 500 milliseconds to ensure a smooth user experience
- **Load Handling:** The system should support at least 10 concurrent users without performance degradation
- **Data Processing Speed:** Calculations related to superposition, entanglement, and decoherence must be completed within 1 second to maintain real-time interactivity
- **Latency:** Server requests should have a maximum latency of 250 milliseconds under normal conditions

4.5.2 Scalability Requirements

- The system must be easily scalable to support increased player loads by leveraging cloud-based or containerized deployment solutions
- Database queries should be optimized to handle a 25% increase in data load without a significant impact on retrieval times
- The system should support modular expansions, allowing for future additions of new quantum mechanics concepts and gameplay modes

4.5.3 Security Requirements

- The game must prevent Cross-Site Scripting (XSS), SQL Injection, and Cross-Site Request Forgery (CSRF) attacks to protect user data

- If user authentication is required, passwords should be hashed using a standard bcrypt algorithm or similar

4.5.4 Usability and Accessibility Requirements

- The user interface must be intuitive, allowing new players to grasp basic gameplay easily
- The game should follow basic accessibility practices, ensuring color contrast and keyboard navigation
- An introductory tutorial should guide players through core quantum concepts interactively

4.5.5 Reliability and Availability Requirements

- The system should aim for 95% uptime, allowing for occasional maintenance and updates
- In case of failure, a manual restart process should be able to restore the system
- A backup system should store player progress and settings, or if they prefer, the player should back up their progress locally

4.5.6 Cross-Browser Compatibility

- The game must be compatible with at least one major browser, such as Google Chrome and Internet Explorer
- It should function consistently on desktop devices, ensuring smooth gameplay across devices

4.5.7 Maintainability and Extensibility

- The codebase should follow MERN stack best practices, with proper documentation and clear folder structures
- The system should allow easy modifications and straightforward updates through modular code
- Unit tests should cover all key functionalities to ensure reliable performance and early detection of bugs

4.6 Domain (Business) Requirements

In order to meet both the educational and market objectives, the these domain requirements must be addressed:

1. Educational Objectives and Target Audience:
   - The system must simplify complex quantum concepts (i.e., superposition, entanglement, and decoherence) to suit a younger audience in quantum computing.
   - Learning outcomes should be clearly defined to ensure that gameplay facilitates gradual learning.
2. Progressive Gameplay Structure:
   - The game is separated into levels representing a stepwise increase in complexity: superposition, entanglement, decoherence.

- Each level must build upon the previous one, ensuring that users develop a solid understanding before advancing to more complex concepts.

3. Regulatory Considerations:

   - The game must comply with relevant educational software standards, including accessibility guidelines and data privacy regulations for minors.
   - The product must adhere to any client-specified success criteria, ensuring accurate representation of their field (quantum).

4. Stakeholder and Client Engagement:

   - Continuous interaction with educators, students, and technical experts is required to validate the educational impact and usability of the game.
   - Feedback loops must be integrated into the development process to ensure that the system aligns with evolving requirements and expectations of all stakeholders.

Chapter 5

System Architecture

– Chan, Martinez, Moks, Ng, Ponte, Quach

5.1 Context Diagram (Level 1)

The system consists of three main parts: Frontend (React), Backend (Node.js/Express), and Database (MongoDB).

5.1.1 Actors & Interactions:

- **Player:** Uses the web-based interface to interact with the game
- **Game Server:** Processes user actions, resolves qubit superposition, and handles game logic
- **Database (MongoDB):** Stores user progress and game state

5.2 Container Diagram (Level 2)

This diagram breaks the system into key components.

- **Frontend (React.js):** Handles UI interactions and displays game elements
- **Backend (Node.js/Express):** Processes game logic, superposition resolution, and API requests
- **Database (MongoDB):** Stores user progress, game data, and qubit states
- **External API (if applicable):** Potential future integration with quantum simulation tools

5.3 Component Diagram (Level 3)

This zooms into the backend, breaking it into smaller components:

- **Game Engine (gameEngine.js):** Handles superposition, entanglement, and decoherence logic
- **Knapsack Manager (knapsack.js):** Tracks player-selected qubits and their properties
- **User Controller (userController.js):** Manages player authentication and progress
- **Database Service (dbService.js):** Connects and queries MongoDB

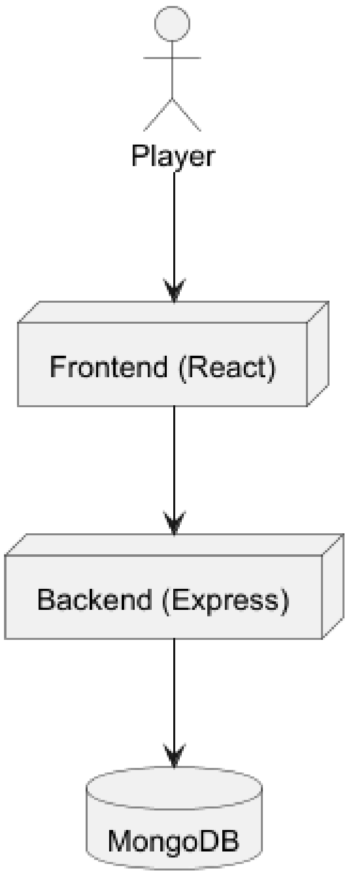

Figure 5.1: Context Diagram: Level 1

5.4 Code Diagram (Level 4)

This is an optional deeper dive into class structures. A simplified UML-like breakdown:

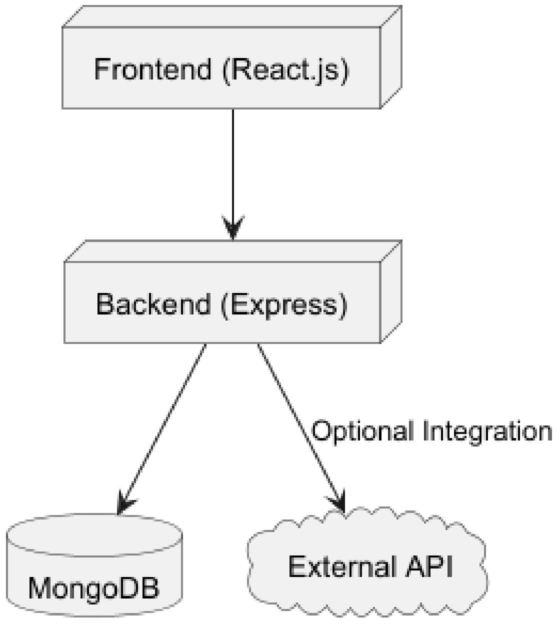

Figure 5.2: Container Diagram: Level 2

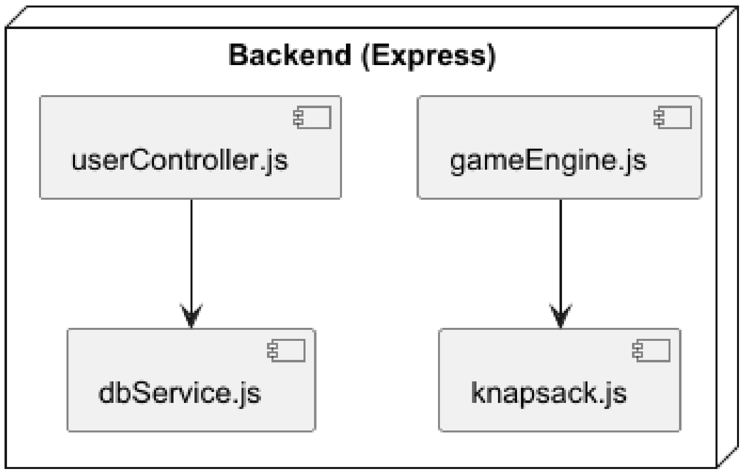

Figure 5.3: Component Diagram: Level 3

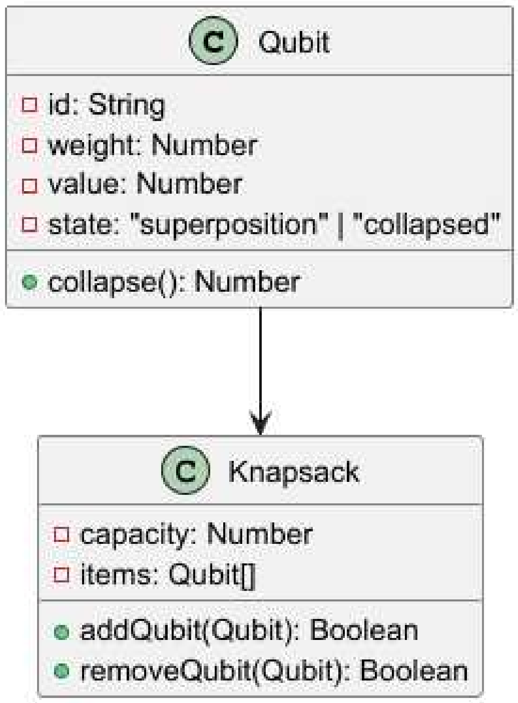

Figure 5.4: Code Diagram: Level 4

Chapter 6

Use Cases

– Chan, Martinez, Moks, Ng, Ponte, Quach

6.1 Table of Use Cases

Table 6.1: Use Cases Table

Use Case	Requirements	Name and Description
UC_1	$REQ_{4.2.1}$	ucQubitSuperposition is a use case describing how a player interacts with qubits in superposition before adding them to the knapsack. The qubits will appear visually unstable until selected, at which point they stabilize.
UC_2	$REQ_{4.2.1}$	ucKnapsackUpdate is a use case describing how the knapsack dynamically updates based on the addition or removal of qubits. The player will always see an updated total value and remaining weight.

6.2 Use Case Diagrams

6.3 Use Case

Table 6.2: . reqQubitSuperposition

Use Case 1 (ucQubitSuperposition) *Experience Qubits in Superposition*
Requirements: $REQ_{4.2.1}$The system must visually represent qubits in a superposition state with flashing colors and changing values until the player selects a qubit.
Diagrams: Figure ucQubitSuperposition (*Figure* 6.1)
Brief description: The system allows players to experience qubits in a superposition state, where they appear visually unstable until added to the knapsack, triggering stabilization.
Primary actors: Player
Secondary actors: None.
Preconditions: 1. The player has started a new round. 2. Qubits are available for selection.
Main flow: 1. The player views qubits on the screen. 2. The qubits appear in an unstable state, flashing colors and varying values. 3. The player selects a qubit and adds it to their knapsack. 4. The selected qubit stabilizes to a single value and color. 5. The system updates the knapsack contents accordingly.
Postconditions: 1. Selected qubits are stabilized. 2. The knapsack's contents are updated.
Alternative flows: 1. A1: If the player deselects a qubit before adding it to the knapsack, the qubit remains in superposition. 2. A2: If the round ends without selecting a qubit, all qubits remain in superposition and are removed.

Table 6.3: .ucKnapsackUpdate

Use Case 2 (ucKnapsackUpdate) *Update Knapsack's Total Value and Weight*
Requirements: $REQ_{4.2.1}$The system must update the knapsack's total value and remaining weight each time a qubit is added or removed.
Diagrams: Figure ucKnapsackUpdate (*Figure* 6.2)
Brief description: The system updates the player's knapsack's total value and remaining weight dynamically whenever a qubit is added or removed.
Primary actors: Player
Secondary actors: None.
Preconditions: 1. The player has an active game session. 2. The knapsack has available capacity.
Main flow: 1. The player selects a qubit and adds it to the knapsack. 2. The system updates the knapsack's total value and remaining weight. 3. The updated information is displayed on the screen.
Postconditions: 1. The total value and remaining weight are correctly updated.
Alternative flows: 1. B1: If the player removes a qubit, the total value and remaining weight adjust accordingly. 2. B2: If adding the qubit exceeds the knapsack's weight limit, an error message is displayed, and the qubit is not added.

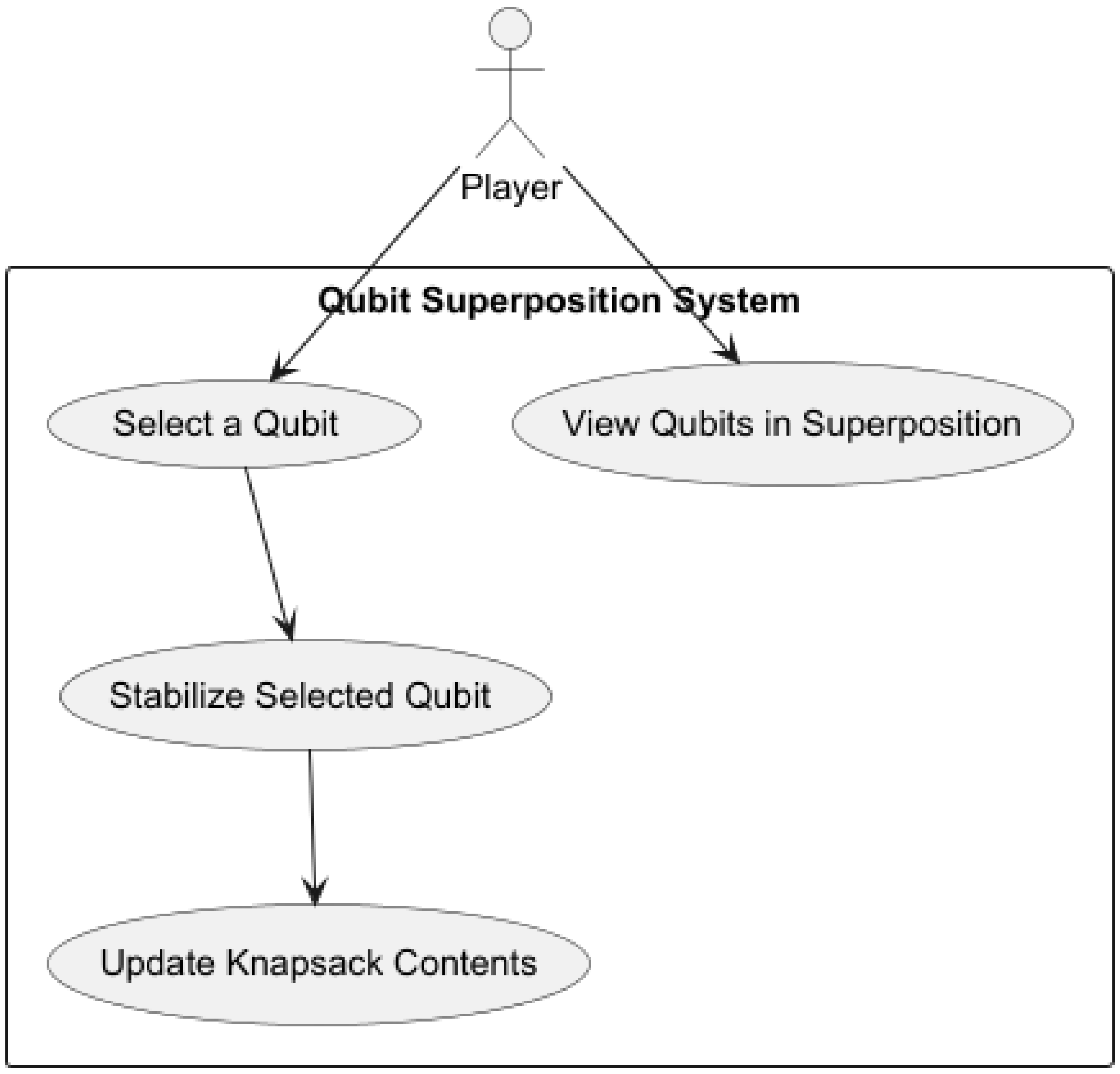

Figure 6.1: : ucQubitSuperposition UC_1.

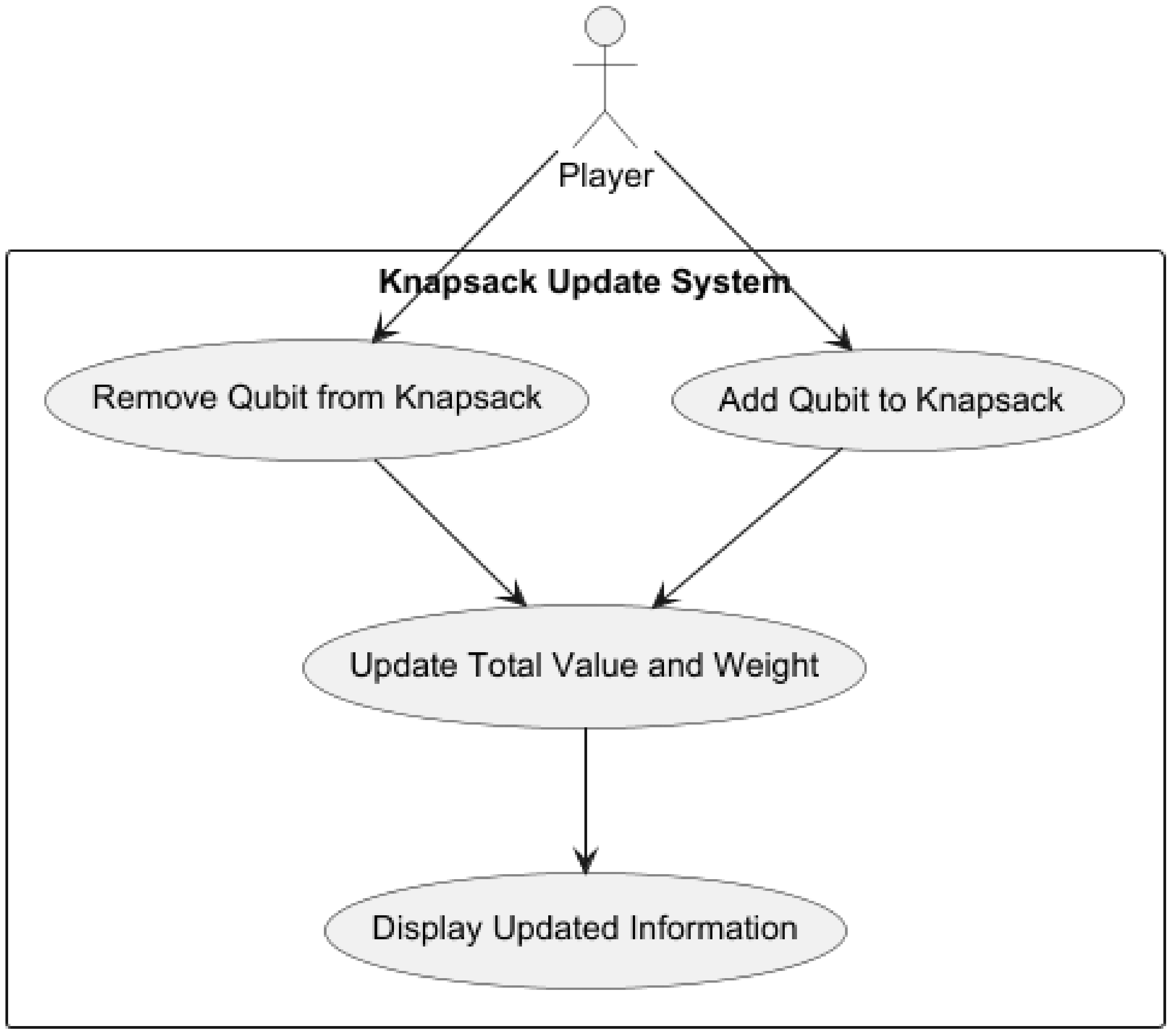
Player
Knapsack Update System
Remove Qubit from Knapsack
Add Qubit to Knapsack
Update Total Value and Weight
Display Updated Information

Figure 6.2: : ucKnapsackUpdate UC_2.

Chapter 7

User Interface Design

– Author Name

7.1 User Persona

Define the persona of your users. You may have more than one user person for capturing different types of user roles for your system.

7.2 User Interface Design

Please provide preliminary paper prototype of your user interface. This should capture the key use cases/ user stories you defined in your project specification. Note that you do not need to cover every corner case. Make sure that your UI design covers the major use cases/stories, and the goals, tasks, and scenarios.

Chapter 8

User Stories

– Quach

8.1 Sprint 1 (27 Feb – 13 Mar)

1. **Set Up MERN Project**
   User Story: As a developer, I want to set up a basic website to serve as a platform for the game. Features will be built on top of this skeleton.

2. **Implement Knapsack Functionality**
   User Story: As a player, I want to select and add qubits to my "knapsack" so that I can maximize my value, without exceeding my weight capacity. If weight exceeds, I want to be notified that it is not possible.

   2.1. **Create Basic "Qubit"**
   User Story: As a developer, I want to create a generic "qubit" object so I can implement more complex features (superposition) on top of this.

   2.2. **Implement Qubit Selection and Deselection**
   User Story: As a player, I want to add qubits to my knapsack and view the changes to my knapsack's total value and remaining weight.

   2.3. **Implement Knapsack Updates After Qubit Selection or Deselection**
   User Story: As a backend developer, I want to adjust the user's total value and remaining capacity after adding/removing a qubit to/from their knapsack. These updated values should be sent back to the frontend in a compact object.

   2.4. **Trigger Error Message If Capacity Exceeded**
   User Story: As a backend developer, I want to check the validity of a player's move before processing it. If adding a qubit causes the knapsack to be overweight, the move will not process, and a message will appear to notify the player.

3. **Implement Superposition in Qubits**
   User Story: As a player, I want to experience quantum foundations by seeing qubits exist in superposition.
   Qubits represent "items" available to be added to my knapsack with capacity n. The weight of a qubit is fixed (random, from 0 to n). The value fluctuates between a static range, and will either fixate to

whichever comes first: (1) the value at the moment I add it to my knapsack, or (2) a random integer after 5s.

4. **Create Landing Page**
   User Story: As a frontend developer, I want to develop a landing page to introduce the game and provide instructions for players so I can ensure that players understand the game objectives and mechanics before they start playing.

5. **Add Button to Finalize Knapsack**
   User Story: As a player, I want to click a button to finalize my knapsack when I am satisfied with my work so I can end the round.

   5.1. **Send Score to Database**
   User Story: As a backend developer, I want to store the players' scores to a database to provide information for the leaderboard feature.

8.2 Sprint 2 (20 Mar – 3 Apr)

1. **Implement Entanglement Between Qubits**
   User Story: As a player, I want to experience quantum entanglement by adding/removing linked qubit pairs.
   When a player adds a qubit to their knapsack, its entangled partner automatically stabilizes in a correlated manner.

   1.1. **Represent Entanglement Visually**
   User Story: As a player, I want to see entanglement represented by a visual line that connects 2+ qubits.

   1.2. **Resolve Entangled Qubits**
   User Story: As a player, I want to see connected qubits resolve to the same value so I can learn about the behavior of entangled qubits.

2. **Implement Decoherence Mechanisms**
   User Story: As a player, I want to learn about decoherence by experiencing random "events" that will decrease the capacity of my knapsack so I can learn to recover and adapt quickly. These events can be represented by a shaky screen, along with a message that the capacity has been reduced.

3. **Write Explanations For Quantum Concepts**
   User Story: As a player, I want to learn about the concept of focus, before each round starts. A description box should show, and when I am ready to start the round, I can hide the box.

8.3 Backlog

- **Check Educational Accuracy**
  User Story: As a backend developer, I want to write tests and confirm with our client to align our expectations so I can to ensure that the randomness of our qubits accurately simulates superposition.
- **Perform Penetration Testing**
  User Story: As a backend developer, I want to simulate cyberattacks on my system to identify vulnerabilities.

Chapter 9

Implementation

– Author Name(s)

9.1 Introduction

The Preliminary Implementation Demo showcases the early development stages of the XXX system. The demo highlights the project's progress by presenting a working prototype (in the initial stages) of the core functionality, focusing on the integration of XXX. This stage serves as a foundation for further refinements, and feature expansions in preparation for the final submission.

The Preliminary Implementation Demo showcases the early development stages of the knapsack system. The demo highlights the project's progress by presenting a working prototype (in the initial stages) of the core functionality, focusing on the integration of the knapsack and qubit objects themselves, as well as the selection/deselection of qubits, updating of the knapsack, and simulating superposition of qubits. This stage serves as a foundation for further refinements, and feature expansions in preparation for the final submission.

9.2 Demo Objectives

The Preliminary Implementation Demo focuses on achieving key milestones in the early development of XXX. The specific goals of the prototype demo include:

1. Validating Key Functionalities
   (a) XXX
2. Establishing Feasibility
   (a) XXX
   (b) XXX
3. Gathering Initial Feedback
   (a) XXX
   (b) XXX

9.3 Prototype Description and Demo Results

XXXXX

9.3.1 Hardware Update

XXXX

9.4 Implementation Details

XXXX

9.5 Challenges Encountered and Solutions

XXXX

9.6 Future Work and Next Steps

As the project progresses, the team will focus on further development and refinement of XXX to meet the goals outlined for the final submission. Key areas of focus include:

9.7 Conclusion

XXXXXXXX

Bibliography

Index