

Programme

Master of Software Engineering 180 Credits

Course

MSE800 Professional Software Engineering

(Level 8, 30 credits)

Assessment 2

Object-Oriented Project

Brain Up: The Learning Adventure

Group H

Student Name: Eunseok Choi - Student ID: 270597067

Student Name: Fabricio Mello Mulato - Student ID: 270516212

TABLE OF CONTENTS

1.	INTRODUCTION	4
2.	TEAM FORMATION.....	5
3.	PROJECT KICK-OFF MEETING	6
4.	THE MĀORI PERSPECTIVE	6
5.	REQUIREMENTS GATHERING MEETINGS	7
6.	PROJECT	7
6.1.	BASIC FEATURES	8
6.2.	WINNING CONDITION	8
6.3.	EXTRA FEATURES.....	8
6.4.	TECHNICAL ASPECTS	9
7.	REQUIREMENT ANALYSIS AND PRIORITISATION	9
8.	AGILE DEVELOPMENT SPRINTS	10
9.	COST ESTIMATION	12
10.	ACCEPTANCE CRITERIA AND TESTING.....	14
11.	REFLECTION REPORT	14
11.1.	LEARNINGS AND IMPROVEMENTS	14
11.2.	CULTURAL CHALLENGES	15
11.3.	TEAMWORK STRATEGIES	16
12.	CONCLUSION	17
	APPENDIX A.....	18
	APPENDIX B.....	21
	APPENDIX C.....	24

FIGURES AND TABLES

Figure 1: Group Logotype	5
Figure 2: Sprints at Jira.....	10
Figure 3: Salary research from Python Developer.....	12
Figure 4: Example of Sprint, Use Stories and Tasks.....	18
Figure 5: Timetable from Jira.....	18
Figure 6: Board from Jira.....	18
Figure 7: Dashboard from Jira	19
Figure 8: Meeting note from Confluence.....	19
Figure 9: Technical Discussions from Confluence	19
Figure 10: Historic commits from both members from GitHub	20
Figure 11: Repository from GitHub.....	20
Figure 12: Class Diagram (UML	24
Figure 13: Use Case Diagram (UML).....	25
Figure 14: Sequence Diagram (UML)	26
Figure 15: Entity-Relationship Diagram.....	27

1. INTRODUCTION

“Brain Up: The Learning Adventure” is an interactive learning game for 9 to 12-year-olds that brings education to life through questions and answers in Science, English, and Māori Vocabulary, in the first version. Designed for two players, the game encourages knowledge growth and friendly competition, helping players build skills for future challenges.

This project utilizes Python to develop a game with a graphical user interface (GUI) using CustomTkinter and Pygame. These libraries are chosen to create an intuitive and user-friendly experience. An SQLite3 database will be used to store players names and ages, questions and game data, such as leaderboards and performance history, ensuring data persistence. GitHub will serve as a central repository for collaborative development.

Following an Agile methodology with Scrum principles, the project was managed using Jira, integrated with Confluence for documentation and meeting notes, ensuring efficient task tracking between the two developers. The management structure is based on Epics, User Stories, Tasks, and Sub-Tasks, organised into 4 one-week sprints to enhance tracking and efficiency, allowing for regular review and adjustment of the project plan to deadline. This approach provides flexibility and continuous adaptation to project needs, with a focus on delivering a high-quality product that meets expectations.

Responsibilities were evenly distributed: one developer will focus on the graphical interface (GUI), while the other will handle database management. Both developers collaborated on the game’s logic, which not only manages the game’s flow and mechanics but also acts as a bridge between the graphical interface and the database, following the Model-View-Controller (MVC) design pattern. MVC separates concerns by dividing the application into three components: the Model (handling data and business logic), the View (managing the user interface), and the Controller (linking the two), ensuring a more organised and maintainable code structure.

2. TEAM FORMATION

The formation of the group for this activity occurred through affinity, culminating in a team of 2 participants who deliberately sought to foster cultural and ethnic diversity. The group comprises one member of South Korean descent and another of Brazilian origin. The name KOBRA was selected as an allusion to the initials of both countries of origin (KOREA and BRAZIL), which in various languages means "snake", evoking the Python programming language, whose emblem is a python snake. We fashioned a logotype for the group, depicting the flags of both nations alongside a snake, referencing the group's name and the technology utilised in the project, thus encouraging engagement and a sense of belonging, as shown in Figure 1.



Figure 1: Group Logotype

This combination of distinct cultural backgrounds furnished varied perspectives throughout the project's advancement, enriching discussions and proposed solutions. Although the team lacks members from the Māori community, we elected to incorporate elements of Māori culture and vocabulary into the game, encompassing questions and answers about this opulent tradition.

This initiative aims to encourage the integration of children from numerous nationalities and respect for local culture, as elaborated in Section 4 of this document. Considering the tenets of Te Tiriti o Waitangi (Participation, Protection and Partnership), despite having no Māori members in the team, we integrated Māori cultural elements into the game's development, acknowledging the significance of this culture in the local context and promoting its appreciation through our product.

3. PROJECT KICK-OFF MEETING

The initial interactions of the KOBRA team occurred during class intervals when preliminary concepts were examined. Subsequently, we conducted a formal gathering via MS Teams video conference to deliberate on the initial project specifications and forthcoming actions.

Ensuing meetings were orchestrated through the Confluence platform, where we documented minutes, queries raised, and determinations reached. This methodology facilitated the maintenance of a methodical and lucid chronicle of all team discussions.

Following the scope definition, we established an environment within JIRA software, integrated with Confluence, to administer the project's Epics, User Stories, Sprints, tasks and schedules, as well as prioritise the backlog for each sprint. This integration afforded a crystalline visualisation of work advancement and enhanced communication amongst team constituents.

4. THE MĀORI PERSPECTIVE

Our game promotes the preservation of Māori language and culture through the principles of Te Tiriti o Waitangi - partnership, participation, and protection. Developed with respect for Te Mana Raraunga and Te Ara Tika guidelines, the application encourages children to explore ancestral traditions and vocabulary in a playful manner whilst incorporating fundamental cultural values. Following the Ngā Tikanga Paihere framework, we ensure ethical and culturally appropriate use of Māori data.

Ngā Tikanga Paihere guiding principles are meant to encourage the use of both ethical and culturally appropriate data that is consistent with the principles of Te Tiriti o Waitangi: partnership, participation, and protection. The principles guarantee that the crucial issues are recognised and that research makes a positive contribution to society through addressing inequalities and promoting inclusiveness.

The educational game aims to connect international learners with Māori children, enabling them to collaboratively explore subjects through interactive activities. By learning vocabulary and customs, players cultivate respect for cultural diversity while forging

intercultural bonds that celebrate shared and distinct identities, thereby bridging global communities. Grounded in these principles, the game serves as a dynamic tool fostering cultural integration, cross-disciplinary learning, and collective well-being.

These principles guided significant decisions, from question content to game mechanics to relationship between members of the Team. The formulated questions were submitted for validation to tutor Aron Kumar, to ascertain whether they contained inaccuracies or might be deemed inappropriate in any manner, to ensure our work remains respectful and precise. The feedback indicated they are appropriate to Māori culture and perspective.

5. REQUIREMENTS GATHERING MEETINGS

We conducted structured offline meetings every Monday and Tuesday, during class intervals, supplemented by flexible quick online sessions Wednesday through Sunday to address emerging tasks.

For game development insights, we gathered feedback from our supervising professor and our children, implementing suggestions to simplify user creation and enhance gameplay experience. We compiled a comprehensive feature list and validated it with stakeholders, including our team members and their children who represented the target user.

Our collaborative approach incorporated diverse perspectives, ensuring cultural sensitivity throughout development. This inclusive methodology helped identify crucial improvements to user creation and gameplay mechanics, making the experience more accessible for our target audience. By valuing cultural differences, user experiences, and adapting to feedback, our development process remained responsive to the needs of a diverse user base.

6. PROJECT

The primary objective of “Brain Up” is to create an engaging and educational game that enhances learning through interactive quizzes. The project aims to:

- Make learning enjoyable and interactive for children aged 9 to 12.
- Encourage strategic thinking and healthy competition through two-player gameplay.

- Provide a structured quiz format with randomized questions, hints, and a time limit to simulate real-world test-taking scenarios.
- Ensure data persistence by storing player progress, scores, and quiz questions in an SQLite database.
- Implement a user-friendly and visually appealing interface using CustomTkinter and Pygame.
- Follow Agile and Scrum methodologies to deliver a high-quality, well-structured, and maintainable application.

6.1. BASIC FEATURES

The MVP (Minimum Viable Product) Gameplay consist in:

- Desktop version – players play together on the same computer.
- Two-player, turn-based system.
- Select two players to start the game. Option to add or delete names.
- Each session consists of 10 multiple-choice questions (4 options per question).
- Questions cover subjects like Science, English, and Māori Vocabulary. More subjects can be added.
- Players have 15 seconds to answer each question by default, but this setting can be adjusted.
- Correct answers add 10 points. Incorrect answers subtract 5 points. If time runs out, points remain unchanged.
- No question repetition within the same session.
- Simple registration (name & age, no login/password).
- Users select answers by clicking on options and using interactive buttons.

6.2. WINNING CONDITION

- The player with the highest score at the end of the session wins.
- Players can use hints and skips strategically, affecting their scores.
- Rankings can be checked to see the top 20 highest-scoring players across all games.

6.3. EXTRA FEATURES

- Hint (Twice per game): Reduces earned points by 50%. Wrong answers are not affected.
- Skip Question (Twice per game): The player can skip a question, and it will return later.

- Bonus Final Question:
 - Correct answer: +20 points (or +10 points if a hint was used).
 - Incorrect answer: -10 points.
- If time runs out, points remain unchanged.
- Configurable Settings: Players can adjust the time limit per question and the number of questions per session.
- Load New Questions via JSON File: Allows easy expansion of the question bank.

6.4. TECHNICAL ASPECTS

This section outlines the technical aspects of the project, detailing the key technologies and tools used in its implementation. Additional information about the code can be found in Appendices A (Screenshots From Jira, Confluence And Github), B (Main Classes And Methods) and C (Diagrams: UML Diagrams and Entity-Relationship).

In addition, the full code is available on the Blackboard platform and can be accessed in the GitHub repository upon request via the following link: <https://github.com/fmulato/Assesment2>.

- Python 3.13.2
- Based on MVC (Model, View and Controller) principles
- Programming Oriented to Object (POO)
- GUI built with CustomTkinter.
- Scores stored in a database (SQLite3).
- New questions released and upload by a JSON file
- Each developer used the IDE they felt most confident with and familiar with, PyCharm and VSCode interchangeably, without affecting the project as code was maintained in GitHub's remote repository.

7. REQUIREMENT ANALYSIS AND PRIORITISATION

Our requirements analysis for “Brain Up” commenced with brainstorming sessions wherein we identified essential functionalities for the educational game. We employed Scrum methodology to administer our project, establishing a product backlog in JIRA with boards arranged in "To Do", "In Progress", and "Done" columns.

Requirement prioritisation was founded on educational value for children of varied backgrounds and feasibility of implementation within the project timeframe. The inclusion of Māori cultural content received elevated priority for both its educational merit and alignment with Te Tiriti o Waitangi principles.

We integrated JIRA with Confluence to document meeting minutes and manage outstanding issues, bug rectifications, and adjustments throughout development. This integration facilitated decision tracking and enhanced our knowledge management.

We instituted a cross-review system whereby, typically, when one member developed a component, the other examined it, ensuring superior quality. Queries regarding unforeseen circumstances arising during development were addressed in Daily Scrum meetings, utilising Confluence or, for more expedient matters, MS Teams.

Pair working, where one member developed whilst another examined, bolstered our communication and product quality, whilst the JIRA Board enabled us to visualise workflow and swiftly identify bottlenecks.

8. AGILE DEVELOPMENT SPRINTS

Our project was structured across 4 sprints, employing agile methodologies to ensure incremental and consistent deliverables, as Timeline presented in Figure 2.

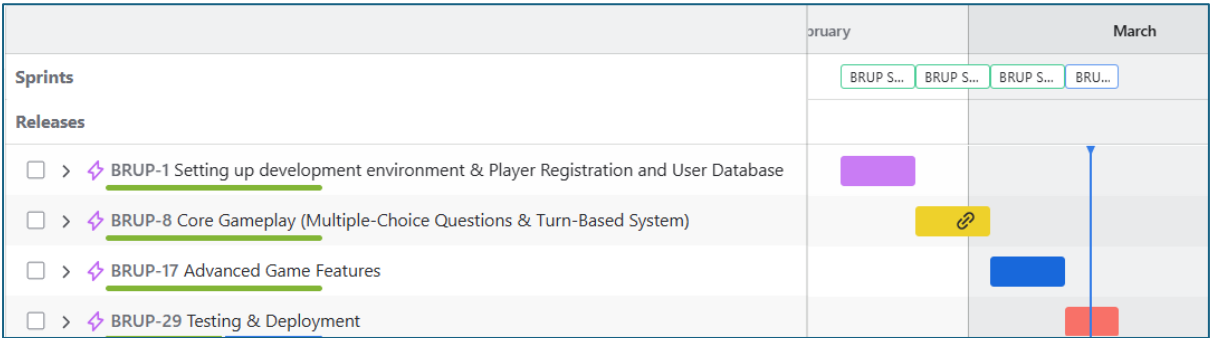


Figure 2: Sprints at Jira

- **Sprint 1: Setting up development environment & Player Registration and User Database**
 - Concentrated on establishing the foundational architecture and implementing core functionalities, including the question-answer framework.
- **Sprint 2: Core Gameplay (Multiple-Choice Questions & Turn-Based System)**

- It was devoted to user interface development and incorporation of initial Māori cultural elements, with consultation of reference materials to guarantee cultural accuracy and respect.
- **Sprint 3: Advanced Game Features**
 - We implemented the scoring system and varying difficulty levels, whilst expanding the question repository with multicultural content, prioritising backlog items according to feedback from previous reviews.
- **Sprint 4: Testing & Deployment**
 - We conducted usability examinations, rectified identified bugs and finalised user experience enhancements, culminating in documentation preparation and product presentation. JIRA utilisation proved essential for progress monitoring, with 15-minute daily meetings for team alignment and impediment removal.

In adherence to Agile and Scrum methodology, Epics were allocated one per sprint. Each Epic contained 2-3 User Stories, with corresponding tasks created for implementation. Tasks were assigned to two developers based on either their expertise or as learning opportunities, with workloads carefully balanced according to initial specialisations: Fabricio managed the graphical interface (GUI) and Eunseok handled the database. Game logic and integration were divided into smaller subtasks between the two developers, balanced by allocating easy, medium, and challenging tasks. The complete breakdown of this structure can be accessed via the following link: <https://mse800-team-kobra.atlassian.net/jira/software/projects/BRUP/boards/3/timeline>.

To illustrate, we've included a complete User Story example below, with tasks arranged in priority sequence, focusing on delivering value to the final user:

- Epic (BRUP-17): Advanced Game Features → Sprint 3
 - User Story (BRUP-19): “As a player, I want to ask for hints twice per game so that I can get help when stuck”
 - Task (BRUP-45): Implement “Ask for Hint” button
 - Task (BRUP-68): Implement field for Hint in database
 - Task (BRUP-69): Implement method to display hint and manage remaining balance
 - Task (BRUP-54): Implement hint penalty in scoring system

- Task (BRUP-70): Implement frame to display each player's remaining balance

Sprint meetings followed standard Scrum structure, consistently considering cultural differences. We incorporated cultural awareness in our interactions, particularly when developing Māori-related content, maintaining Te Tiriti o Waitangi principles as a reference for our decisions.

Pair working strengthened our communication and product quality, whilst the JIRA Board enabled us to visualise workflow and swiftly identify bottlenecks.

9. COST ESTIMATION

This cost estimation details development expenses for labour, tools, and resources, using JIRA story points as a reference. Two developers worked on the project for four weeks, with a structured daily effort. The hourly rate was estimated using salary comparison data from Glassdoor (<https://www.glassdoor.co.nz/>) for Python Developers, as shown in Figure 3.

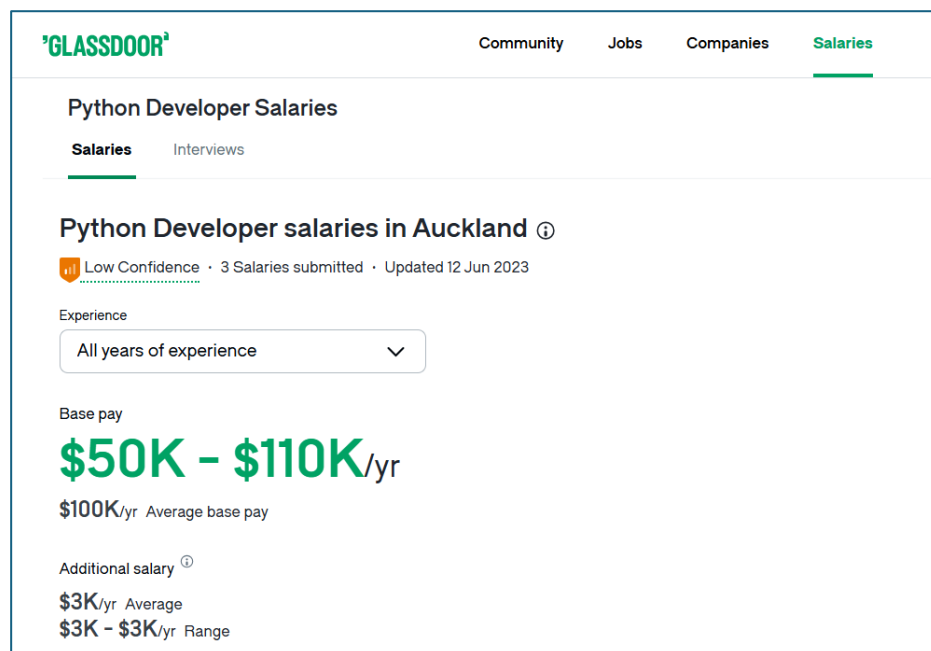


Figure 3: Salary research from Python Developer

$$\text{Average} = (\$ 50,000 + \$ 110,000) / 2 = \$ 80,000$$

$$\text{Hourly rate} = (\$ 80,000 + \$ 3,000 / 52 \text{ weeks}) / 40 \text{ hours per week} = 39.90 \sim 40.00$$

Table 1: Hourly Rate per developer

Developer	Role	Hourly Rate	Participation Rate
-----------	------	-------------	--------------------

Eunseok	Junior Developer	\$ 40.00	100%
Fabricio	Junior Developer	\$ 40.00	100%

Development Duration & Work Hours

- Start Date: February 17, 2025
- End Date: March 14, 2025
- Total Duration: 4 weeks (28 days)
- Work Hours per Day: 2 hours per developer
- Total Work Hours per Developer: 2 hours/day \times 5 days/week \times 4 weeks = 40 hours
- Total Work Hours (Both Developers): 40 hours \times 2 = 80 hours

Table 2: Cost Breakdown

Cost Category	Description	Calculation	Total
Labour Costs	Eunseok (Junior)	\$ 40.00 x 40 hours	\$ 1,600
	Fabricio (Junior)	\$ 40.00 x 40 hours	\$ 1,600
Tool Costs	JIRA (Project Management)	Free	\$0
	Confluence (Documentation)	Free	\$0
Total Cost			\$ 3,200
Markup	Profit margin	100%	\$ 3,200
Subtotal			\$ 6,400
GST	Goods and Services Tax	15%	\$ 960
Income Tax	IRD	28% over profit	\$ 896
Final Price			\$ 8,256

The total estimated development cost for the project is \$ 8,256. No additional tool costs were incurred, as JIRA and Confluence were used without charge. Each developer worked two hours per day for four weeks, summarizing 40 hours per developer. This estimate accurately reflects the labour investment, including profit margin and tax, assuming 100% developer capacity.

Final Summary:

- Service Cost: NZ\$ 3,200
- Profit Before Tax: NZ\$ 3,200
- Price Excluding Taxes: NZ\$ 6,400
- GST (15%): NZ\$ 960
- Final Price for the Customer (incl. GST): NZ\$ 8,256
- Corporate Tax (28% on profit): NZ\$ 896 (paid by the company on profit)
- Net Profit After Tax: NZ\$ 3,200 (3,200 - 896)

10.ACCEPTANCE CRITERIA AND TESTING

To ensure “Brain Up” meets all functional, technical, and user experience requirements, the project follows a structured acceptance criteria and testing process. Each user story is evaluated based on predefined acceptance criteria in JIRA, ensuring that all implemented features align with project goals.

The testing process includes:

- Unit Testing – Validates individual components, including game logic, database interactions, and UI functionality.
- Integration Testing – Ensures seamless interaction between the GUI, quiz logic, and database.
- User Acceptance Testing (UAT) – Conducted with a small group of players (our children), assessing usability, engagement, and cultural inclusivity.
- Performance Testing – Evaluates system responsiveness, including UI performance and database query efficiency.

JIRA was used to track testing progress, with test cases and issue tracking linked to corresponding user stories. Confluence was used to document testing strategies, results, and refinements, ensuring clear communication and traceability throughout development.

Additionally, testing considered cultural inclusivity, ensuring that the game remains engaging, respectful, and appropriate for Māori communities.

By following this comprehensive testing and tracking approach, “Brain Up” ensures a high-quality, user-friendly, and culturally inclusive learning experience for all players.

11.REFLECTION REPORT

11.1. LEARNINGS AND IMPROVEMENTS

Throughout the development of "Brain Up: The Learning Adventure", our team acquired and enhanced various technical skills. Implementing a Python-based educational game using CustomTkinter and Pygame for the GUI provided valuable experience in creating interactive applications. Working with SQLite3 for database management allowed us to

develop robust data persistence mechanisms for storing player information, questions, and game statistics.

The adoption of the Model-View-Controller (MVC) design pattern proved instrumental in maintaining a clean separation of concerns, making our codebase more organized and maintainable. This approach facilitated effective collaboration, with one developer focusing on the graphical interface while the other managed database operations, with both contributing to the game logic that bridged these components.

The development of this project provided valuable learnings across various technical domains, while also identifying opportunities for continuous improvement.

Key Learnings:

- In-depth understanding of GUI concepts, including positioning, control, and enabling/disabling widgets such as labels, entries, and buttons.
- Separation of the user interface from business logic and the database, requiring enhanced understanding of object-oriented programming principles.
- SQLite3 database management, including the creation and initial population of tables and data, existence checks, and integration with Python.
- Practical experience in team collaboration using GitHub, including identifying changes and employing commands such as merge, commit, pull, push, and stage. Also learning how to update project with the most recent version of the code.
- Manipulation of JSON files integrated with the SQLite3 database.
- Introduction and development of intermediate proficiency in project management tools like JIRA and Confluence, including their integration.

Areas for Technical Enhancement:

- Implementation of more advanced graphical and animation techniques to enhance visual appeal for the young audience.
- Creation of optimised database queries for improved performance, particularly as the question bank expands.
- Exploration of more sophisticated algorithms for question selection, aiming for more precise adaptation to individual player knowledge levels.

11.2. CULTURAL CHALLENGES

In Integrating Māori cultural elements provided both valuable learning opportunities and significant challenges. As a team composed of South Korean and Brazilian members, we

lacked direct knowledge of Māori traditions, requiring careful research and validation. Key challenges included ensuring linguistic accuracy, balancing cultural representation, avoiding misrepresentation, and applying Te Tiriti o Waitangi principles. To address these issues, we consulted tutor Aron Kumar, and adhered to frameworks like Ngā Tikanga Paihere and Te Ara Tika, highlighting the importance of cultural consultation in educational content development.

One major difficulty was aligning team schedules due to cultural differences, family commitments, and the challenge of balancing personal and academic responsibilities. Although both team members lived in Auckland, their daily routines and family dynamics differed, influenced by their cultural backgrounds and the demands of raising young children.

These differences required flexibility, compromise, and mutual understanding to ensure effective collaboration. Clear communication and structured planning played a crucial role in overcoming these challenges while fostering a respectful and inclusive team environment.

11.3. TEAMWORK STRATEGIES

Our teamwork approach leveraged Agile methodology with Scrum principles, providing structure while maintaining flexibility. The integration of Jira with Confluence created a comprehensive project management environment that enhanced communication and task tracking. Our decision to distribute responsibilities based on strengths—GUI development and database management—while collaborating on game logic proved efficient.

Effective teamwork strategies included:

- Structured offline meetings on Mondays and Tuesdays complemented by flexible online sessions throughout the week
- Cross-review system where team members examined each other's work, enhancing quality and knowledge sharing
- Daily Scrum meetings to address emerging issues and maintain alignment
- Pair programming sessions for complex components, improving code quality and team communication

Areas for teamwork development include:

- Establishing more formalized processes for cultural validation earlier in the project lifecycle
- Developing better mechanisms for balancing workload during intensive development periods

- Creating more comprehensive documentation protocols for knowledge transfer
- Implementing more structured user testing sessions with representatives from the target audience

12. CONCLUSION

The development of “Brain Up: The Learning Adventure” provided valuable lessons in technical implementation, cultural sensitivity, and teamwork. We believe we have successfully created an educational game that incorporates elements from various cultures while leveraging advanced features of the Python and SQL programming languages. This approach enabled the practical application of essential functionalities, ensuring the quality and effectiveness of the final product.


Beyond the technical aspects, the project offered a valuable learning experience in project management, allowing us to utilise tools that will remain relevant throughout our professional careers. Working as a team presented challenges and insights distinct from individual work, requiring interpersonal skills, organisation, and commitment. The process also highlighted both our existing technical competencies and the areas in which we need to further develop our expertise.

A fundamental aspect of this project was our commitment to the principles of Te Tiriti o Waitangi, which transformed what could have been a simple quiz game into a more meaningful educational tool that respects and celebrates cultural diversity. This approach not only enriched the final product but also deepened our understanding of the importance of cultural diversity and its value in an educational context.

Thus, “Brain Up: The Learning Adventure” not only fulfilled its educational purpose but also served as a practical experience in technical development and professional collaboration, strengthening skills that will be essential for our future academic and professional challenges.

APPENDIX A

SCREENSHOTS FROM JIRA, CONFLUENCE AND GITHUB



BRUP Sprint 3 - Advanced Feat. 3 Mar – 9 Mar (12 issues)		0	0	5	Complete sprint	...
<input type="checkbox"/> BRUP-18	As a player, I want to skip a question twice per game so that I can avoid difficult questions strategically.	ADVANCED GAME FEA...	IN PROGRESS	-		
<input checked="" type="checkbox"/> BRUP-44	Implement 'Skip Question' functionality	ADVANCED GAME FEA...	IN PROGRESS	-		EC
<input type="checkbox"/> BRUP-19	As a player, I want to ask for hints twice per game so that I can get help when stuck.	ADVANCED GAME FEA...	IN PROGRESS	-		
<input checked="" type="checkbox"/> BRUP-46	Implement Hint Functionality to Reduce Answer Choices	ADVANCED GAME FEA...	IN PROGRESS	-		EC
<input checked="" type="checkbox"/> BRUP-47	Implement hint penalty in scoring system	ADVANCED GAME FEA...	IN PROGRESS	-		EC
<input checked="" type="checkbox"/> BRUP-49	Implement database storage for final game results	ADVANCED GAME FEA...	IN PROGRESS	-		EC
<input checked="" type="checkbox"/> BRUP-50	Implement button to show rules and details on the first screen	ADVANCED GAME FEA...	DONE	3		FM
<input checked="" type="checkbox"/> BRUP-48	Implement bonus/penalty points for the last question	ADVANCED GAME FEA...	DONE	2		FM

Figure 4: Example of Sprint, Use Stories and Tasks

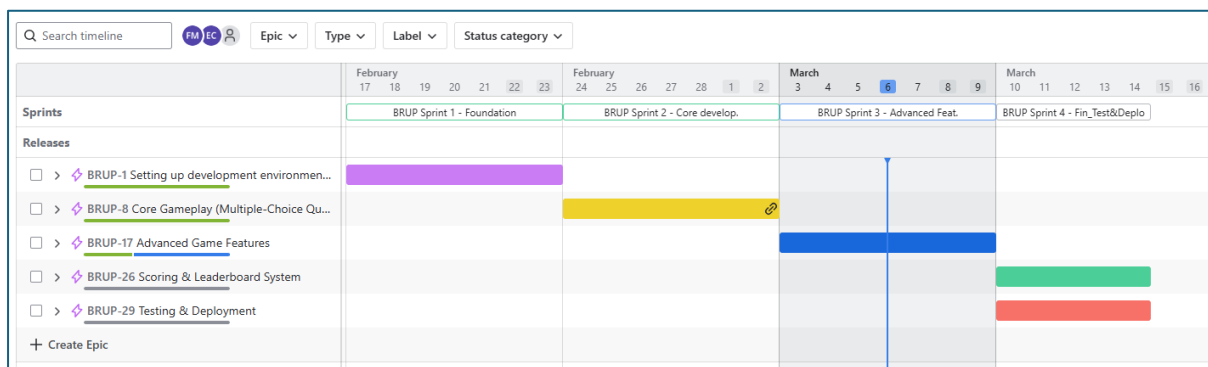


Figure 5: Timetable from Jira

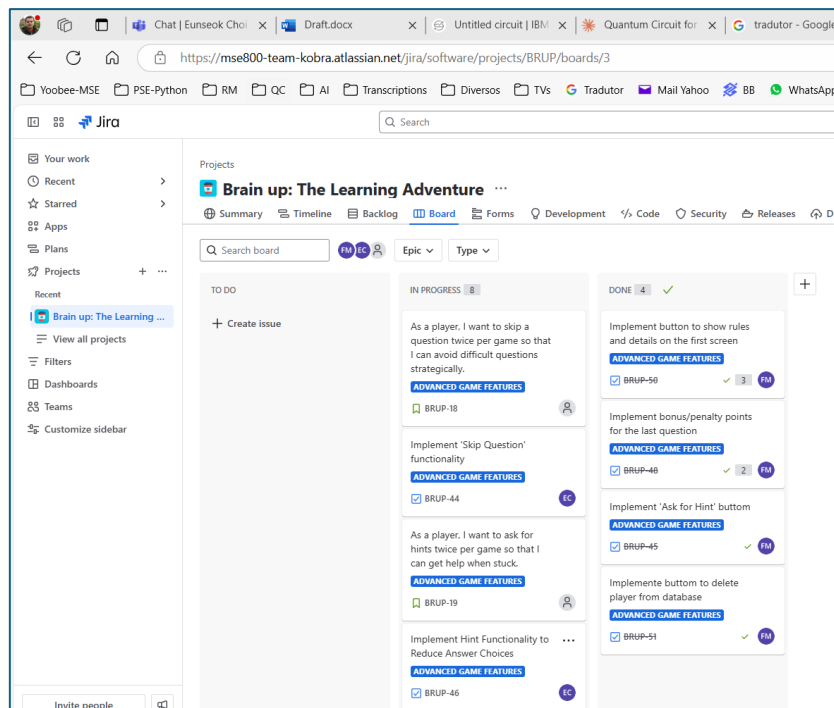


Figure 6: Board from Jira

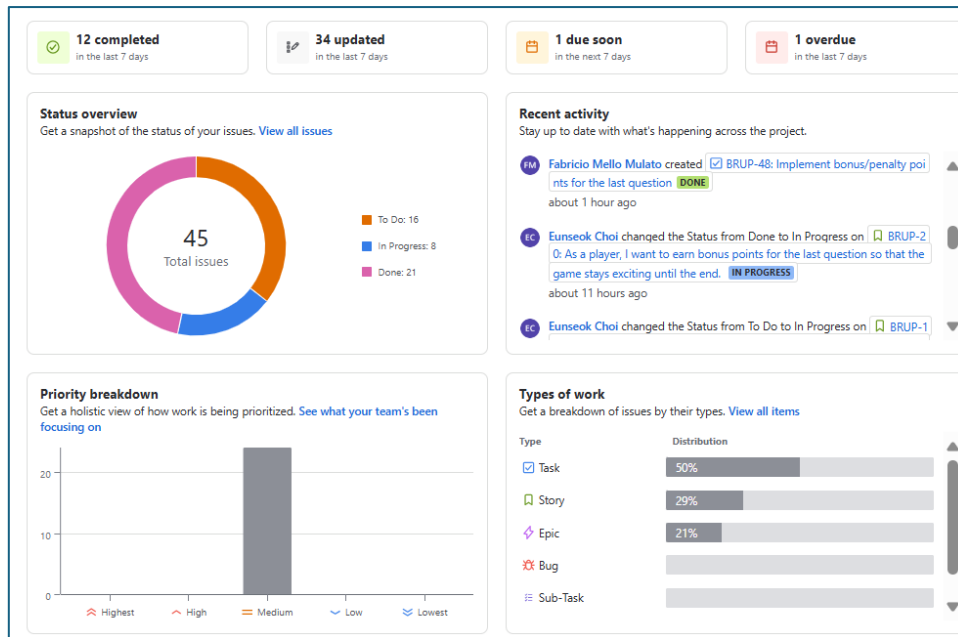


Figure 7: Dashboard from Jira

Meeting Note - 20 Feb, 2025 (2)

Owned by [Fabricio Mello Mulato](#) ***
Last updated: Feb 20, 2025 • 2 people viewed

Date
Feb 20, 2025

Participants

- @Eunseok Choi
- @Fabricio Mello Mulato

Goals

- Validate Layout
- Define GUI basic configurations

Discussion topics

Item	Notes	Etc
Screen width and height	parameterizable	
Divisions in frames	5 fixed frames	
Font setup	parameterizable	
Background colours	parameterizable	

Decisions

- We defined 3 main frames (left, center, right), the central one being subdivided into top, center and bottom. 5 frames in total. More flexibility
- Configurable screen width and height, centering on the screen regardless of user configuration
- Font and background colours parameterizable

Figure 8: Meeting note from Confluence

2 page comments

Eunseok Choi
Feb 23, 2025

I agree with you. Since we are aiming for a simple game, it would be better to store all tables in a single database. I will create the quiz table under users.db.

Now that users.db stores not only user data but also questions, we can either keep the name users.db or rename it to something more appropriate. what about 'main.db'?

[Reply](#) • [Edit](#) • [Delete](#)

Fabricio Mello Mulato
Feb 23, 2025

I guess we can rename to brainup.db, as it is the game's general database.

[Reply](#) • [Edit](#) • [Delete](#)

Figure 9: Technical Discussions from Confluence

February 12, 2025 – March 12, 2025

Period: 1 month ▾

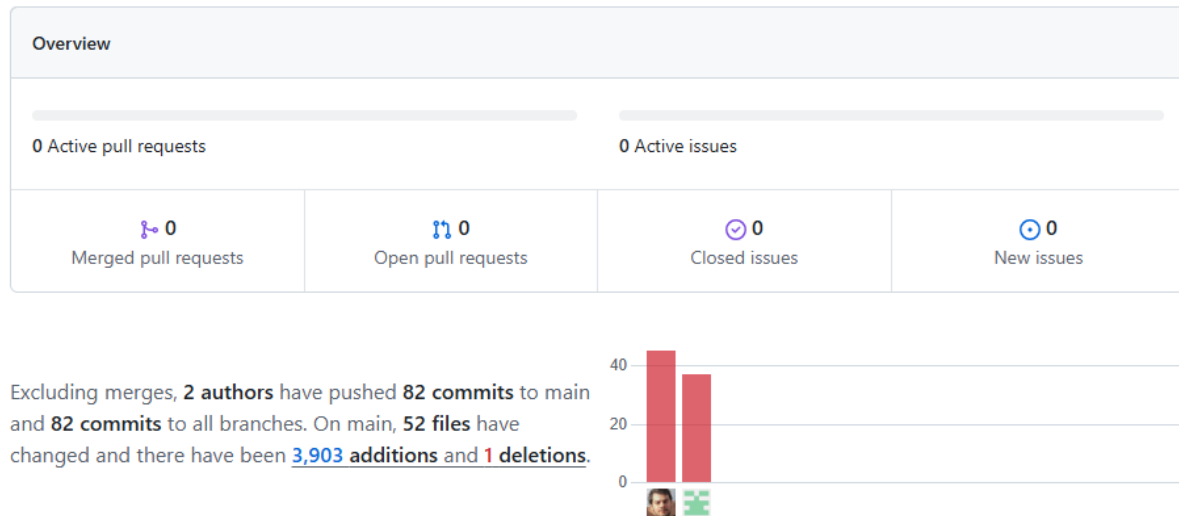


Figure 10: Historic commits from both members from GitHub

The screenshot shows the GitHub repository interface for 'Assesment2'. The left sidebar displays the file explorer with the following structure:

- main (selected)
- .idea
- docs
- src
 - .idea
 - __pycache__
 - prototypes
 - bonus.mp3
 - buzz.mp3
 - database_management.py
 - gui.py
 - main.py (selected)
 - questions.json
 - questions_math.json
 - quiz_logic.py
 - sql_statement.py
 - test_utils.py
 - tic-tac.wav
 - utils.py

The main content area shows the code editor for 'main.py'. The code is as follows:

```
1 """
2 This is the main file. It is responsible for starting the application by
3 calling the StartScreen class from the gui module.
4 The database is loaded from a JSON file and the questions are inserted
5 into the database.
6 """
7 import gui
8 from database_management import DataBase
9
10 def app():
11     """Start the application."""
12     DataBase().load_db_from_json("questions.json")
13     gui.StartScreen()
14
15 if __name__ == "__main__":
16     """Start the application."""
17     app()
```

Figure 11: Repository from GitHub

APPENDIX B

MAIN CLASSES AND METHODS

1. main.py: The main file responsible for starting the application by calling the StartScreen class from the gui module and calling database_management to create DB.

2. database_management.py

Class DataBase: Manages user data, players, and quiz questions in a SQLite database.

Methods and Functions:

- `__init__`: Initialises the class and creates the tables in the database.
 - `create_tables`: Creates tables for players, scores, and questions if they do not exist.
 - `register_user`: Adds a new user to the database with a unique username and date of birth.
 - `save_age_to_scores`: Saves the user's age in the scores table using the player's ID.
 - `get_all_players`: Retrieves all players and their latest recorded ages from the scores table.
 - `load_db_from_json`: Loads questions into the database from a JSON file.
 - `check_if_question_exists`: Checks if a question already exists in the database.
 - `load_questions`: Fetches all questions from the database.
 - `get_ranking`: Retrieves the ranking of players from the database.
 - `update_setup`: Updates the settings in the database.
 - `delete_player`: Deletes a player from the database.
 - `save_final_score`: Saves the player's final score in the database with their age.
-

3. gui.py

Class StartScreen: The first window where players are selected, managing player registration, settings, and starting the game.

Methods and Functions:

- `__init__`: Initialises the start screen and sets up the graphical interface.
- `exit`: Exits the game.
- `update_new_questions`: Updates the questions from a JSON file.
- `display_player_buttons`: Displays the player buttons on the screen.
- `toggle_player_selection`: Manages player selection, allowing a maximum of two.
- `update_button_colors`: Updates the colours of the player buttons based on selection.
- `update_start_button`: Enables the start button when two players are selected.
- `add_new_name`: Opens the dialog to add a new player.
- `delete_player`: Deletes the selected player from the database and updates the player list.

- `show_ranking`: Displays the ranking of players.
- `show_rules`: Displays the rules of the game.
- `open_setup`: Opens the settings dialog.
- `start_game`: Starts the game with the selected players.

Class GameScreen: The main window where the quiz is played, managing the display of questions, options, and results.

Methods and Functions:

- `__init__`: Initialises the game screen and sets up the graphical interface.
- `run`: Executes the graphical interface.
- `enable_submit_button`: Enables the submit button.
- `get_current_player_name`: Returns the name of the current player.
- `countdown`: Starts the countdown timer.
- `exit`: Exits the game.
- `back_to_start_screen`: Returns to the start screen to select new players.

Class AddNameDialog: Pop-up dialog for entering the name and date of birth of a new player.

Methods and Functions:

- `__init__`: Initialises the dialog and sets up the graphical interface.
- `save_birthday`: Saves the selected date of birth in the format "YYYY-MM-DD".
- `apply`: Saves the input values when the OK button is pressed.

Class ShowRulesDialog: Pop-up dialog that displays the rules of the game.

Methods and Functions:

- `__init__`: Initialises the dialog and sets up the graphical interface.

Class ShowRankingDialog: Pop-up dialog that displays the ranking of players.

Methods and Functions:

- `__init__`: Initialises the dialog and sets up the graphical interface.

Class SetupDialog: Pop-up dialog for game setup options.

Methods and Functions:

- `__init__`: Initialises the dialog and sets up the graphical interface.
 - `save_settings`: Saves the settings and updates the global settings in the database.
-

4. quiz_logic.py

- **Class: Logic:** Contains the logic for the quiz game, managing the display of questions, options, and results.
 - **Methods and Functions:**
 - `__init__`: Initialises the game logic and sets up the current player and time limit.
 - `start_timer`: Starts the timer.
 - `stop_timer`: Stops the timer.
 - `display_question`: Displays the next question and options.
 - `play_last_question_sound`: Plays a sound indicating it is the last question.
 - `next_turn`: Moves to the next player's turn and updates skip and hint buttons.
 - `is_last_round`: Checks if the current question is one of the last two.
 - `check_answer`: Checks the selected answer and applies hint penalties if a hint was used.
 - `next_question`: Moves to the next question.
 - `determine_quiz_winner`: Determines the winner of the quiz and saves the final scores.
 - `underline_current_player`: Highlights the current player.
 - `hint`: Allows the current player to use a hint up to two times per game.
 - `skip`: Allows the current player to skip a question up to two times per game.
-

5. sql_statement.py: Contains all SQL statements for creating and managing the database, organising queries for better maintenance.

- **Statements:** Creation of tables, insertions, selections, updates, and deletions of data in the database (queries).
-

6. utils.py

Class Utils: Contains utility functions for the game, such as selecting random elements and shuffling answers.

Methods and Functions:

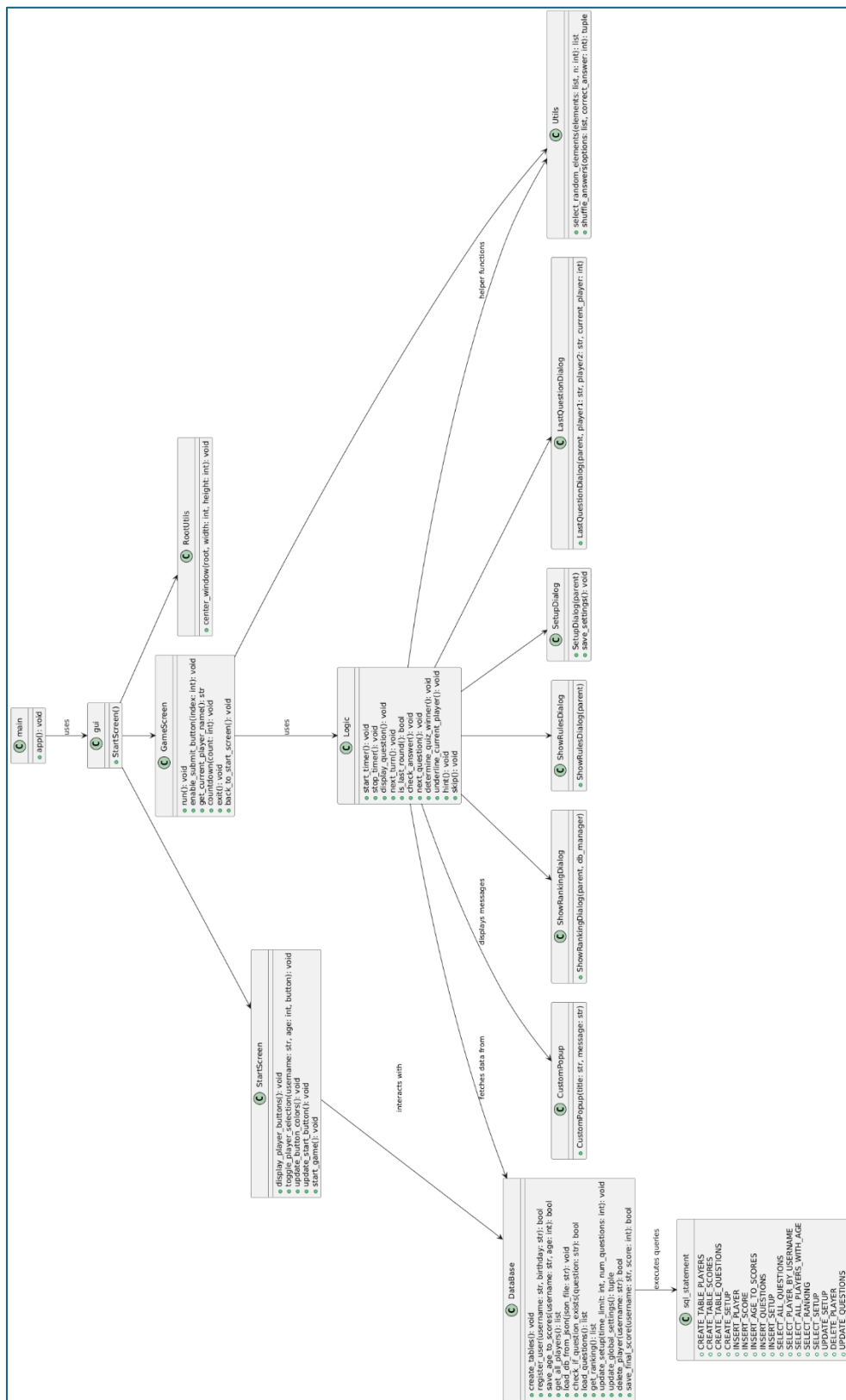
- `select_random_elements`: Selects random elements from a list and returns their indices and values.
- `shuffle_answers`: Shuffles the answers and returns a new list with the correct answer index updated.

Class CustomPopup: Class for creating custom pop-up windows for success and error messages.

Methods and Functions:

- `__init__`: Initialises the pop-up window with a title and message.

Diagrams: UML Diagrams and Entity-Relationship



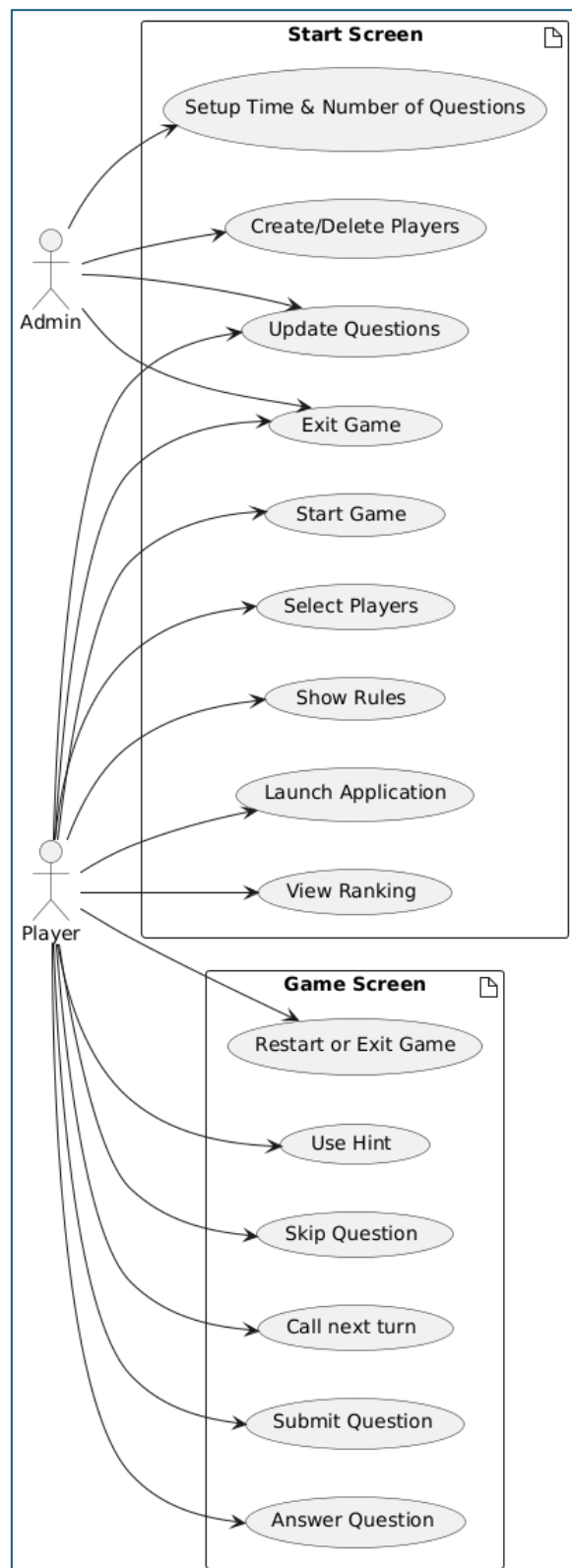


Figure 13: Use Case Diagram (UML)

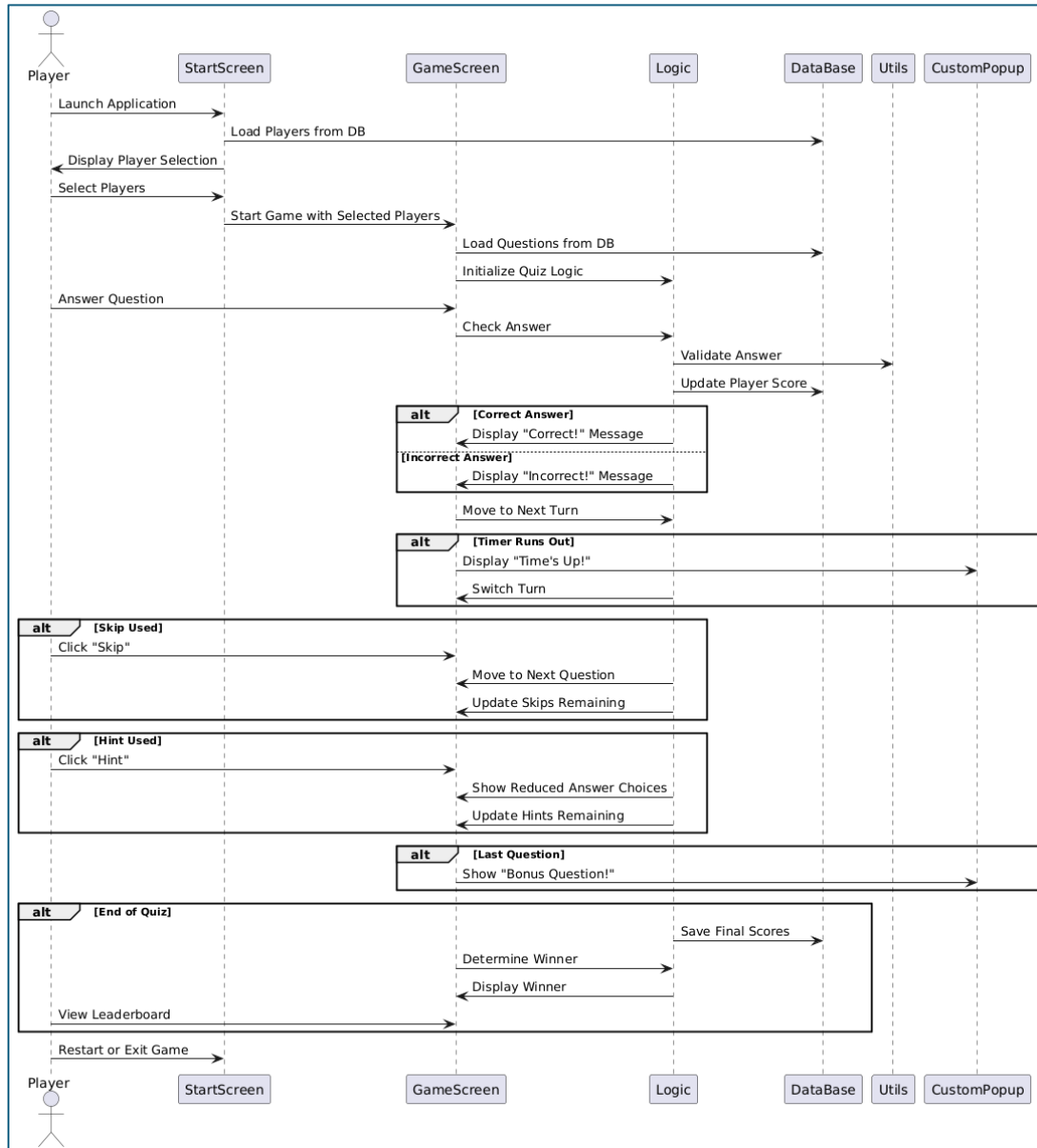


Figure 14: Sequence Diagram (UML)

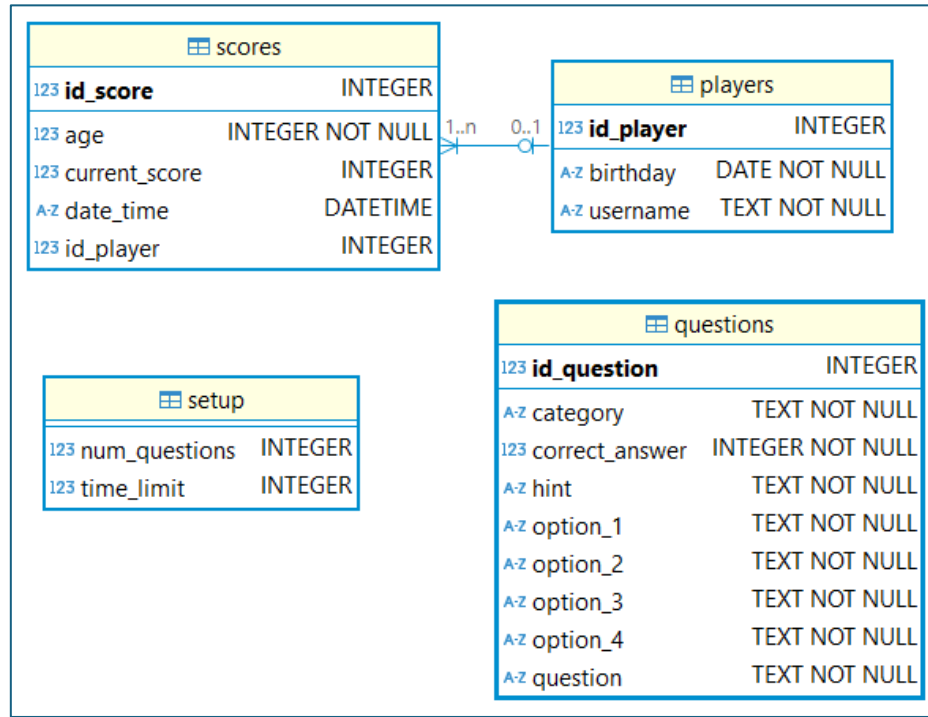


Figure 15: Entity-Relationship Diagram