PART 2 TEAM REPORT

Authors

DARRYL NG BING BING (2303161)

PHILICIA LEE FEI LI (2303359)

TOH ZHEN WEI (2303006)

LIM JING YU (2303001)

CHERYL LEE JIAYU (2303278)

SHEILA SOH LI EN (2303134)

LAB-P11-4

Module: INF1009 Object-Oriented Programming

Date: 29/03/2024

Table of Contents

1. Introduction to Healthy Heroes

This report will describe the overall game system design for the game, Healthy Heroes, an engaging game developed using IntelliJ, Java, and libGDX, where players embark on a heroic quest to promote wellness through an exhilarating gaming experience. The game leverages OOP concepts such as Encapsulation, Inheritance, Abstraction, Polymorphism, Aggregation, Composition, Dependency, Association, and the SOLID principles to ensure flexibility, extensibility, and ease of experimentation. In this action-packed adventure, players (i.e. children) navigate their valiant heroes through a torrential downpour of healthy and unhealthy foods, strategically collecting nutritious options while avoiding tempting treats. Triumph is achieved by collecting five healthy foods to unleash frenzy mode, doubling points, and reinforcing the importance of proper nutrition. With its immersive design and transformative potential, Healthy Heroes offers a thrilling adventure that not only entertains but also inspires players to embrace healthy living.

2. System Design

The system design follows a modular architecture, incorporating clearly defined components and layers to ensure efficiency and maintainability.

2.1 Game Engine Layer

- Scene Manager:
    - The “SceneManager" class, responsible for orchestrating the creation, transition, and disposal of scenes, belongs to the Game Engine Layer. It manages the presentation and flow of the game by handling scene rendering and transitions.
    - Subclasses of the “Scene” class (e.g., EndScene, PlayScene, HelpScene, MenuScene, SettingsScene, LeaderboardScene, Tips, ObjectiveScene, and UsernameInputScene) also belong to the Game Engine Layer as they are primarily concerned with scene rendering and user interface elements. Each scene is encapsulated within a subclass of “Scene”, enabling specialised logic and functionality tailored to its purpose.

- Input / Output Manager:
    - The Input / Output Manager, responsible for managing player interactions and providing feedback within the game, belongs to the Game Engine Layer. It handles various input sources, such as keyboard and mouse clicks, and manages visual and auditory feedback.
    - For instance, translating player inputs into actions within the game environment, in the “PlayScene”, pressing left or right arrow keys prompts the player character to move left or right, respectively. Moreover, it utilises event listeners, such as “Clicked” to “Changed”, to detect mouse interactions, enabling actions like menu selection and object interaction.
    - An example of auditory feedback through sound effects will be when encountering a detrimental element like a “BadItem”, the Manager triggers a corresponding sound effect to alert players to the negative consequence.

2.2 Game Logic Layer

- Simulation Lifecycle Manager:

    - The Simulation Lifecycle Manager, responsible for overseeing critical processes such as resource initialization and continuous game state updates, belongs to the Game Logic Layer. It manages the underlying processes essential for the simulation's operation.
    - For instance, during resource initialization, essential resources like game assets and data structures are set to their initial states, laying the groundwork for the simulation. This ensures that the simulation starts with a stable foundation, optimising memory usage and system stability.

- Entity Manager:
    - The "EntityManager" class, responsible for managing dynamic entity interactions and collision detection, belongs to the Game Logic Layer. It manages entity creation, updates, and interactions, focusing on the core gameplay mechanics.
    - The manager collaborates closely with the "Entity" class and its subclasses (e.g., Player, GoodItem, BadItem), ensures consistent behaviour and interaction among game entities.
    - The "EntityManager" performs several key functions within the game environment. It dynamically updates entity states through its "update(float deltaTime)" method, ensuring responsiveness by incorporating movement based on user input and collision checks. Additionally, the "spawnEntities()" method introduces new entities into the game, adjusting spawn rates and types for balanced gameplay. The "setFrenzyMode(boolean frenzyMode)" method toggles an intensified gameplay mode, enhancing player engagement. Crucial collision and overlap checks are conducted through methods like "checkOverlapping(Entity entity)", ensuring accurate responses to collisions. Furthermore, the "createEntities()" method initialises the game state with a player entity and manages player-specific operations, such as UI element updates. Lastly, the "EntityManager" handles visual aspects like background changes in response to game mode alterations, contributing to player immersion and comprehension of the game state.

- Collision Manager:
    - The "CollisionManager" class, responsible for managing dynamic entity interactions and collision detection, belongs to the Game Logic Layer. It governs the behaviour of entities in response to collisions and ensures the realism and immersion of gameplay.
    - This class exemplifies the application of Object-Oriented Programming (OOP) principles, enhancing the gaming experience through dynamic and realistic entity interactions. Central to its functionality is encapsulation, as evidenced by methods managing collision detection internally while presenting a simple interface. Through polymorphism, the "CollisionManager" seamlessly handles different types of entities, simplifying code maintenance and enhancing extensibility. Inheritance facilitates a hierarchical relationship between entities, ensuring consistency and reducing redundancy. Abstraction layers the game logic, making the system easier to understand and modify.

- AI Control Manager:
    - The AI Control Manager, responsible for governing the behaviour of non-player characters, belongs to the Game Logic Layer. It manages the behaviour of AI-controlled entities, such as good and bad items, to create engaging gameplay experiences.
    - The manager oversees the behaviour of non-player characters, including good items representing healthy food choices and bad items representing unhealthy options. Additionally, visual feedback, such as a progress bar, informs players of their proximity to entering or exiting Frenzy Mode.

- Player Control Manager:
    - The Player Control Manager, responsible for governing the underlying game logic and functionality. It manages player-specific functionalities and interactions, such as movement, health management, and visual representations, which are fundamental aspects of the game's logic and mechanics.
    - The manager encapsulates player-specific functionalities, including keyboard input processing, health management, and visual representation, ensuring a seamless interaction with the game environment. It dynamically responds to player commands through keyboard inputs, enabling effective navigation and engagement with gameplay elements. Moreover, it manages the player's health, tracks the collection of "good items", and handles visual changes in response to game events or achievements. Through encapsulation, polymorphism, and inheritance, the Player Control Manager creates a comprehensive framework for player control and interaction, emphasising the significance of OOP principles in building a cohesive game architecture.

Overall, the Game Engine Layer primarily deals with the presentation and user interaction aspects of the game, while the Game Logic Layer focuses on implementing the core gameplay mechanics and rules that govern the game's behaviour. Each layer plays a crucial role in ensuring the overall functionality and immersion of the game.

The system design follows a modular architecture with clearly defined components and layers. At the core of the architecture lies the Game Engine Layer, which includes components such as Scene, Player, Pause, Play, and Settings. These components manage various aspects of the game, including scene transitions, player movement, gameplay logic, and settings configuration. Additionally, the system employs a SceneManager to handle the management of different scenes and facilitate smooth transitions between them. Each scene is responsible for rendering its elements, updating game logic, and disposing of resources when necessary. The Game Layer encompasses the entity classes like Player and EntityManager, which handles the creation, updating, and rendering of in-game entities. Overall, the system design promotes modularity, separation of concerns, and ease of maintenance, allowing for scalable development and efficient management of game components.

2.3 Creational Design Pattern

In “Healthy Heroes,” our creational design strategically incorporates the Singleton and Factory Method patterns to refine resource management and facilitate dynamic entity creation. These patterns synergize to elevate the architectural robustness and gameplay experience, promoting efficiency and expandability.

2.3.1 Singleton Method

The Singleton pattern is employed through the “SoundManager” class, ensuring a singular, globally accessible instance that manages all sound-related functionalities within the game. This approach centralises sound management, providing consistent audio experiences across various game states and activities.

- Justification:
  Opting for the Singleton pattern for sound management minimises the risk of redundant instances and optimises resource utilisation. This is crucial for maintaining game performance and ensuring a seamless audio environment, which enhances the player's immersive experience.

- Implementation:
  The “SoundManager” guarantees only one instance is created and accessible globally, managing background music, sound effects, and dynamic volume adjustments. This setup fosters a controlled and cohesive sound management system across the game.

2.3.2 Factory Method

The Factory pattern is implemented within the “EntityManager” to dynamically create game entities like items and obstacles. It allows the game to instantiate objects at runtime based on the gameplay requirements, facilitating a modular approach to entity management.

- Justification:
  By utilising the Factory Method pattern, “EntityManager” enhances the game's flexibility and scalability. This design supports easy incorporation of new entity types without altering the core game logic, adhering to the open-closed principle and promoting loose coupling.

- Implementation:
  “EntityManager” leverages factory methods to instantiate entities, adjusting to the game's evolving states and scenarios. This method empowers the game to introduce varied gameplay elements effortlessly, ensuring a dynamic and engaging player experience.

The integration of Singleton and Factory Method patterns in "Healthy Heroes" not only streamlines sound management and entity creation but also aligns with best practices in software design. This dual-pattern approach ensures the game's architecture is both efficient

and scalable, facilitating a robust foundation for current functionalities and future expansions. Through this strategic application, "Healthy Heroes" achieves a balanced and immersive gaming environment, ready to adapt and grow with the players' evolving interests and challenges.

2.4 Improvements to the Engine

In our latest update, the game engine has undergone significant enhancements to elevate the player's experience beyond its previous iteration. A central feature of this new design is the introduction of Frenzy Mode. This mode not only transforms the game's background ambiance, creating a more engaging environment, but also guarantees the drop of only beneficial items, thus amplifying both the game's appeal and the challenge it presents during this period. Once players successfully collect five beneficial items, they unlock Frenzy Mode, which accelerates the game's pace through a rapid succession of advantageous items, presenting an exhilarating and time-limited challenge.

Additionally, to make the game even more attractive, especially to younger audiences, we've enriched the gameplay with improved animations, enhanced visual and sound components. These enhancements are designed to captivate children's attention, making the game more immersive and enjoyable.

Furthermore, we've revised the item creation process. The original design used a singleton pattern, which has now been replaced with an abstract factory pattern. This shift allows for more flexible and scalable item generation, contributing to a more dynamic and varied gaming experience. Coupled with the integration of both positive and negative feedback loops, these advancements aim to promote healthy gaming habits, ensuring that players receive immediate and impactful feedback on their gaming behaviours.

Together, these improvements represent a comprehensive effort to refine the game's mechanics, aesthetics, and overall user experience, making it more engaging and rewarding for players.

3. Object-Oriented Principles

3.1 Encapsulation

Encapsulation is demonstrated in various classes by hiding the internal implementation details and exposing only necessary functionalities through public methods. For instance, in the Player class, the movement logic is encapsulated within the handleInput() method, hiding the implementation details from other classes and ensuring that the player's movement can only be controlled through defined input methods.

3.2 Abstraction

Abstraction is evident in the Scene class, which defines common properties and methods for managing scenes in the game. This abstraction allows different types of scenes to be treated uniformly through a common interface, simplifying scene management and promoting code reuse.

3.3 Inheritance

Inheritance is utilised in the system, as seen in the PlayScene class, which inherits from the Scene class. This inheritance allows the PlayScene class to inherit common scene management functionalities from the Scene class, reducing code duplication and ensuring consistency across different scene implementations.

3.4 Polymorphism

Polymorphism in our system is vividly illustrated through the “Scene” class, an abstract entity that enables varied scene implementations while sharing a common interface. By declaring methods such as “create”, “update”, “render”, and “dispose” as abstract, the “Scene” class mandates their implementation in subclasses, allowing each scene to have unique behaviors under these uniform method signatures. This setup empowers the system to handle scenes of differing functionalities—be it a menu, game level, or settings page—through a single reference type, thus showcasing polymorphism's role in promoting code flexibility and extensibility. Each subclass can have its distinct rendition of these methods, adapting to specific scene requirements while adhering to a consistent framework defined by the “Scene” class.

3.5 Association

The association is observed in the SceneManager class, which maintains a stack of scenes and controls the transition between them. This association allows the SceneManager to interact with different scene objects and manage their lifecycle, facilitating modular and reusable code.

3.6 Aggregation

Aggregation can be seen in the PlayScene class, which contains an instance of the EntityManager class to manage entities within the scene. This aggregation allows the PlayScene class to compose complex behaviour from simpler components, promoting code modularity and reusability.

3.7 Composition

UsernameInputScene has a composition relationship with GameMaster, suggesting that this scene is responsible for capturing and possibly storing user input (username) in a manner that's critical to the game's state managed by GameMaster. The existence of UsernameInputScene is closely tied to the game's state control, indicated by GameMaster. Thus, When the container is destroyed, so are its components, signifying ownership

3.8 Dependency

Dependency is crucial for establishing relationships between classes and ensuring that changes in one class do not adversely affect others. Throughout the system, classes depend on each other to accomplish specific tasks, such as scene management, input handling, and game logic. For example, the PlayScene class depends on the SceneManager class to transition between scenes, highlighting the interdependence between different components of the system.

Our game uses these dependencies as they are essential for various aspects of game development, including rendering graphics, handling user input, managing audio, and storing game data. Each library serves a specific purpose in facilitating different aspects of game development, making them crucial for building robust and feature-rich games using the LibGDX framework.

- com.badlogic.gdx: This is the core library of the LibGDX game development framework. It provides essential features for game development, including graphics rendering, input handling, audio management, and file I/O.
- com.badlogic.gdx.graphics: This library specifically deals with graphics rendering in the LibGDX framework. It includes classes and methods for rendering sprites, textures, shapes, and other graphical elements.
- com.badlogic.gdx.scenes.scene2d: This library provides a scene graph-based UI framework for building user interfaces in LibGDX games. It includes classes for creating and managing UI components such as buttons, labels, tables, and stages.
- com.badlogic.gdx.utils: This library contains utility classes and data structures commonly used in game development. It includes classes for managing arrays, pools, timers, and other utility functions.
- com.badlogic.gdx.math: This library offers mathematical utilities for game development, such as vector and matrix operations, interpolation, and random number generation. It is used for various calculations, including collision detection and resolution.
- java.util: This is a standard Java library that provides a set of common data structures and utility classes. It includes classes like ArrayList, HashMap, and Collections, which are widely used for managing data in Java programs.
- com.badlogic.gdx.input: This library provides classes for handling input events, such as keyboard, mouse, and touch events, in the LibGDX framework. It includes interfaces and adapters for processing input from different sources.
- com.badlogic.gdx.inputprocessor: This library offers classes for processing input events using the LibGDX framework. It includes interfaces and adapters for implementing custom input processing logic, such as gesture recognition and multi-touch handling.
- com.badlogic.gdx.audio: This library deals with playing and managing audio in the LibGDX framework. It includes classes for loading and playing sound effects, music tracks, and other audio assets.
- com.badlogic.gdx.preference: This library provides classes for managing game preferences and other persistent data in a cross-platform way using the LibGDX framework. It allows storing and retrieving user settings, high scores, and other persistent data.

4. Engine-Game Code Separation & Adherence to Software Design Principles

4.1 Engine-Game Code Separation

Our ‘GameMaster’ class serves as the central hub for game management. ‘GameMaster’ handles essential tasks like scorekeeping, Frenzy Mode activation, and resource disposal, while also coordinating with other components such as the scene manager, entity manager, and frenzy manager. Each component is responsible for distinct gameplay aspects, such as scene transitions, entity behaviours, and Frenzy Mode functionality. By leveraging the LibGDX framework and extending the ‘ApplicationAdapter’ class, we harness the engine’s capabilities while incorporating custom game logic within ‘GameMaster’. This approach streamlines maintenance, enhances modularity, and facilitates efficient project development and improvement.

4.2 Software Design Principles: (SOLID)

In our game, the SOLID principles are exemplified to varying degrees across different classes and sub-classes, contributing to a well-structured and maintainable codebase.

4.2.1 Single Responsibility Principle (SRP):

This principle is evident in several classes. For example, the “CollisionManager” class is responsible for handling collisions between entities, ensuring that it focuses solely on this specific task without entangling unrelated concerns. Similarly, the “Userdata” class adheres to this principle by encapsulating functionality related to user data management, such as calculating grades based on scores, keeping its responsibilities clear and focused.

4.2.2 Open / Closed Principle (OCP):

This principle is applied in “ItemFactory” class as it utilises a factory pattern, this class remains open for extension such as creating new types of items, but closed for modification as its existing code does not change when new item types are added. Thus, it promotes flexibility and scalability. Additionally, the “SceneManager” class, through its use of a stack to manage scenes, allows for easy extension by adding new scene types without modifying the existing implementation.

4.2.3 Liskov Substitution Principle (LSP):

This principle ensures that subclasses can be substituted for their base classes without affecting the correctness of the program. In our game, the “GoodItem” and “BadItem” classes demonstrate this principle. Both subclasses inherit from the base class “Item” and can be used interchangeably where an Item is expected, allowing for polymorphic behaviour while maintaining correctness.

4.2.4 Interface Segregation Principle (ISP):

In the “SoundManager” class, segregating sound-related functionalities into distinct interfaces, such as one for volume control and another for sound playback, would help prevent clients from depending on interfaces they do not use, thus promoting a more modular design.

4.2.5 Dependency Inversion Principle (DIP):

This principle encourages decoupling high-level modules from low-level implementation details by depending on abstractions rather than concrete implementations. In our game, the “EntityManager” class interacts with abstractions like “Scene” and “GameMaster”, rather than concrete implementations, promoting flexibility and easier maintenance. Similarly, the “SceneManager” class depends on the abstract “Scene” class, allowing for easy substitution of different scene types without modifying “SceneManager” itself, thus adhering to this principle.

5. Unique and Innovative Features

5.1 Educational Dialogue, Tips, and Learning Objectives

A key feature in the game's development is the thoughtfully crafted Help Scene, which represents an innovative stride in interactive guidance. This scene succinctly informs players about the game mechanics in a way that parallels real-life virtues, such as time management and knowledge acquisition. The instructions provided are tailored to be age-appropriate and easily comprehensible, assuring that players can grasp the gameplay quickly. This direct and interactive approach does more than simply set the stage for the game; it subtly imparts lessons on good habits, such as the value of punctuality and learning, which are represented by the 'good' and 'bad' items within the game. In doing so, the game naturally leads players to make informed choices that align with positive life skills, thus integrating a layer of educational value seamlessly within the fun of gameplay. Moreover, the game incorporates a “Tips” scene before gameplay to share insights into healthy eating habits by providing information on both good and bad foods. By integrating such educational elements into the gaming experience, the game not only entertains but also contributes to the player’s knowledge and awareness of real-world health education. Additionally, a learning objectives scene is included to explain the game’s purpose, highlighting the importance of eating healthy foods for good health. It encourages players to collect nutritious items and avoid unhealthy ones during gameplay.

5.2 Feedback System

The feedback system integrated into the game provides an innovative feature to reinforce educational content through interactive and engaging mechanisms. By employing positive feedback loops, such as granting shields for healthy choices, the game provides immediate effects that validate and encourage good decision-making, effectively motivating players to maintain healthy behaviours. Conversely, negative feedback mechanisms, such as penalties

for selecting unhealthy items, offers a gentle yet impactful learning curve, sensitively guiding players to understand the consequences of poor dietary choices without detracting from the game's enjoyment. This innovative approach not only fosters challenging and immersive gameplay experience but also serves as a powerful tool for instilling healthy eating habits. Through this feedback system, the game effectively communicates the tangible benefits of healthy choices, empowering players to make informed decisions that can positively impact their real-world dietary behaviours and overall well-being.

5.3 Grading System

In the leaderboard scene, Healthy Heroes introduces a dynamic grading system to motivate players further and provide feedback on their performance. This grading system assigns players a grade from A to F based on their scores, offering a clear indication of their progress and proficiency in the game. Players achieving high scores and demonstrating consistent healthy eating habits will earn top grades, while those struggling to make nutritious choices may receive lower grades. By incorporating this grading system into the leaderboard scene, Healthy Heroes not only encourages healthy competition among players but also reinforces the importance of making positive dietary choices. This feature adds depth to the gameplay experience, motivating players to strive for excellence while fostering a sense of accomplishment and pride in their achievements.

5.4 Player Customisation and Voice Record Functionality

Before gameplay, players have the ability to personalise their gaming experience by allowing them to insert their names, adding a touch of individuality to their journey. Additionally, the game incorporates a unique voice record function that enables players to add their own sound effects into the gameplay, enhancing immersion and creativity. This innovative feature provides players with a new level of engagement and customisation, making each gameplay session truly unique and memorable.

6. Learning Outcome

6.1 Incorporation of Educational Elements

In the Part 2 development phase of our game, significant strides were made in embedding educational elements to promote healthy eating habits among kids. The introduction of an educational dialogue or help screen before gameplay, serves as an innovative approach to engage young players in learning about the consequences of unhealthy eating. This proactive step not only sets the stage for the gaming experience but also aligns the game's objectives with essential real-world health education. By highlighting the health risks associated with the consumption of unhealthy foods, we provide a foundational understanding that reinforces the game's underlying message. This educational layer is crucial for fostering informed decision-making among children regarding their dietary choices, thereby extending the game's impact beyond mere entertainment to serve as a valuable educational tool.

6.2 Enhancement of Gameplay Mechanics

The enhancement of gameplay mechanics in Part 2 has been instrumental in reinforcing the game's educational message while ensuring an engaging and enjoyable experience for kids. The implementation of Frenzy Mode introduces dynamic elements that challenge players' reflexes and decision-making under pressure, emphasising the benefits of choosing healthy items. As players collect good items, a Frenzy Mode bar below the score gradually fills up. Once filled, players enter Frenzy Mode, where the scene changes dynamically, adding an extra layer of excitement and challenge to the game. In Frenzy Mode, players earn double points for collecting good items, which enhances the gameplay experience. Thus, making it more enjoyable for players and further emphasising the game's mission of promoting healthy living.

Additionally, before players begin their adventure, the game offers a unique "Record Voice" function. This feature allows players to record their own sound effects, such as cheers or funny noises, which are then integrated into the game. This personalization not only adds an element of fun and creativity but also fosters a deeper connection between players and the game world. By allowing players to incorporate their voices into the game, Healthy Heroes creates a more immersive and engaging experience, encouraging players to become active participants in the healthy eating narrative.

The incorporation of positive and negative feedback loops directly links game actions to the concept of consequences. For instance, granting shields such as an advantage for collecting healthy choices by collecting good items encourages positive behaviour, while penalties for selecting unhealthy options subtly teach the importance of avoiding detrimental habits. These gameplay mechanics are thoughtfully designed to mirror real-life choices and consequences, thereby reinforcing the educational theme of healthy eating. Additionally, the careful refinement of object interaction and lifecycle management ensures a seamless and intuitive gameplay experience, further engaging players in the game's healthy eating narrative. Through these enhancements, the game not only captivates children's interest but also imparts crucial lessons on health and nutrition, making it a valuable tool in the promotion of healthy eating habits among kids.

7. Team Contribution

Our team contribution is shown as follows:

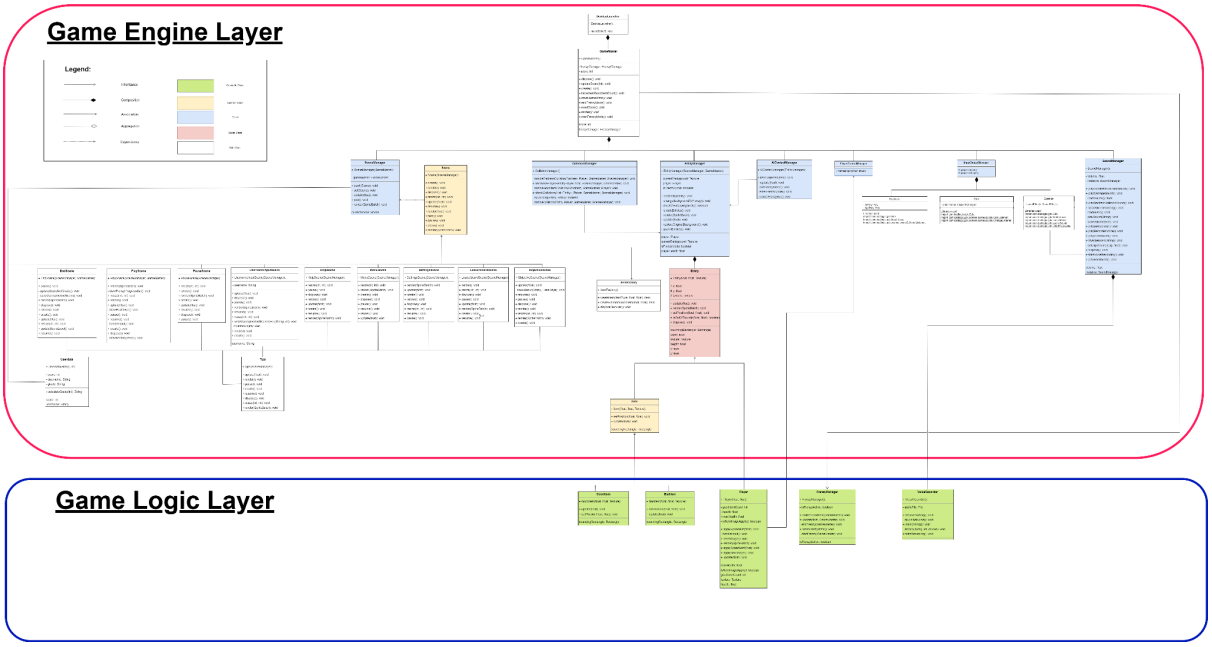
Name	Contribution
DARRYL NG BING BING (2303161)	Responsible for the implementation of input/output management and voice recorder component of the game, improve overall code, UML design
PHILICIA LEE FEI LI (2303359)	Responsible for the implementation of simulation life cycle management component of the game, UML design
TOH ZHEN WEI (2303006)	Responsible for the implementation of AI management, collision management, entity management, frenzy management component of

	the game, overall look & feel of the game and improve overall code
LIM JING YU (2303001)	Responsible for the implementation of scene management, collision management, leaderboard scene, input/output management component of the game, improve overall code, UML design
CHERYL LEE JIAYU (2303278)	Responsible for the implementation of player control, AI control and scene management component of the game, improve overall code, UML design
SHEILA SOH LI EN (2303134)	Responsible for the implementation of entity management component of the game, UML design
The entire team collaborated closely to implement the Game Layer in the game and establish the relationships between different classes, documenting them through a UML Diagram and report. Each team member contributed to both composing the report and creating presentation slides.	

8. Conclusion

In conclusion, our game engine not only provides a solid foundation for a diverse range of games but also shines brightly in our innovative Healthy Heroes venture. Through the seamless integration of object-oriented programming (OOP) principles, our game engine ensures that Healthy Heroes is not just entertaining but also educational. By leveraging OOP concepts, such as encapsulation and inheritance, our game engine facilitates the creation of a dynamic and engaging gameplay experience centred around promoting healthy eating habits among children. From the exhilarating frenzy mode to the strategic decision-making elements, Healthy Heroes harnesses the power of gaming to inspire young minds to make nutritious choices while having fun. With a robust and reusable design, our game engine empowers developers to continue expanding the Healthy Heroes universe, spreading the message of wellness and balanced nutrition to children worldwide.

9. UML Diagram



https://drive.google.com/file/d/1-ABtCVuBUd1xNwqRHqDLHAbAzc1DnLpb/view?usp=sharing