ITWM5043-Software Development

Final Assessment Section: MC-O31

NAME	MATRIC NO	MOBILE NO
Kugendran Chettiarr	**MC240930903**	0127316674

ABSTRACT

The Animal Kingdom simulation project was developed to create a virtual environment that accurately models and observes the behaviors of various animal species. The primary objective was to design and implement specific behaviors for different animal types within this simulated ecosystem, following established guidelines and rules. These behaviors were implemented by defining distinct subclasses derived from a common superclass named Critter.

This methodology utilized key principles of object-oriented programming, particularly inheritance and polymorphism, to achieve a modular and scalable code architecture. Inheritance enabled the reuse of shared functionalities provided by the Critter superclass, while polymorphism allowed each subclass to implement unique behaviors and attributes specific to its respective animal type.

The report outlines the project's objectives, which included exploring object-oriented design principles, adhering to industry-standard programming practices, and developing an interactive and dynamic simulation. It provides a comprehensive analysis of the design strategies adopted, the step-by-step implementation of animal behaviors, and the outcomes observed through rigorous testing. The results validate the effectiveness of the chosen design approach, demonstrating the ability to replicate complex interactions and movements within the simulated ecosystem.

INTRODUCTION

The Animal Kingdom project provides a comprehensive exploration of class inheritance and object-oriented programming principles. It was conceived to offer hands-on experience with these concepts through the development of a simulated ecosystem. At the heart of the project lies the Critter superclass, which serves as the foundational framework for all other classes. This superclass defines fundamental behaviors and attributes shared by all critters, enabling the creation of specialized subclasses with distinct characteristics.

To fulfill the project requirements, five unique subclasses were developed: Bear, Tiger, WhiteTiger, Giant, and NinjaCat. Each subclass was designed to demonstrate specific behaviors, distinct color attributes, and movement patterns tailored to its unique traits. These behaviors and characteristics adhered to a predefined set of rules, ensuring that each critter's actions and interactions contributed cohesively to the simulation.

The primary goal of the project was to programmatically model animal-like behaviors, with a strong emphasis on modularity. This approach allowed individual components of the simulation to operate independently while seamlessly integrating within the broader ecosystem. The focus on reusability ensured efficient sharing of common functionalities across subclasses, reducing redundancy and enhancing code maintainability. Strict adherence to the project specifications underscored the importance of meeting defined requirements while ensuring consistency and precision.

The project also emphasized the practical application of coding skills in Java, highlighting the ability to write structured, maintainable, and extensible code. Additionally, it involved integrating multiple components within an existing simulation framework, reinforcing the significance of compatibility and cohesive design.

Overall, the Animal Kingdom project served as a platform for exploring advanced programming concepts, fostering creativity in designing dynamic and interactive critter behaviors, and demonstrating the practical utility of object-oriented design principles.

PROJECT DESCRIPTION

The project operates within a simulation environment where various Critter subclasses interact. The primary functionalities of each critter subclass are determined by overriding three methods:

1. getMove(CritterInfo info): Defines the action taken by the critter during its turn.
2. getColor(): Specifies the color of the critter.
3. toString(): Provides a string representation of the critter for visualization.

The following five subclasses were implemented:

- Bear: Alternates between / and \ symbols and follows simple movement logic.
- Tiger: Cycles through random colors and behaves dynamically in response to its surroundings.
- WhiteTiger: A specialized Tiger with additional infection-related behavior.
- Giant: Displays a cycling sequence of strings and prioritizes infection or wallhugging movement.
- NinjaCat: Implements unique movement and infection strategies with distinct visuals.

The supporting framework files (CritterMain, CritterModel, CritterPanel) manage simulation setup, rendering, and interactions between critters.

PROJECT DESIGN

Class Descriptions

- Critter (Superclass): Defines default behavior for critters and provides a base for subclasses to override.
- Bear: Alternates display between / and \ every move. Moves by infecting enemies, hopping forward, or turning left. Returns Color.WHITE for polar bears and Color.BLACK otherwise.
- Tiger: Cycles colors (RED, GREEN, BLUE) every three moves. Displays as TGR. Prioritizes infecting enemies, avoids walls, and moves dynamically. - WhiteTiger: Behaves like a Tiger but is always white (Color.WHITE). Changes its string from tgr to TGR upon infecting another critter.
- Giant: Displays a sequence of fee, fie, foe, fum in six-move intervals. Infects enemies in front, hops forward if possible, and turns right otherwise. - NinjaCat: Displays z initially and Z after infecting another critter. Dynamically changes color based on infection status. Implements custom movement logic sensitive to its environment.

PROGRAM WORKFLOW AND LOGIC

1. Initializatin:

- CritterMain creates instances of each critter subclass and initializes the simulation world.
- Each critter is added to the simulation grid, which is enclosed by walls.

2. Simulation Cycle:

- The framework calls the getMove method for each critter to determine its action (hop, turn, infect).
- Critters interact based on their position and the surrounding environment.

3. Rendering:

- CritterPanel visualizes critters using their string representation and color attributes.
- Debug mode displays directional arrows for critters.

DEVELOPMENT DETAILS

Key Implementation Points

- Bear: Tracks its movements using an internal counter to alternate its appearance. Maintains a boolean attribute to distinguish between polar and non-polar bears.
- Tiger: Utilizes a randomization mechanism for color selection, resetting every three moves. Adapts its movement strategy to minimize clustering and avoid obstacles like walls.
- WhiteTiger: Monitors its infection state internally to dynamically update its string representation when necessary.
- Giant: Employs a modular counter to cycle through its text representations at predetermined intervals.
- NinjaCat: Implements custom logic for infection and movement, dynamically adjusting its appearance based on contextual factors.

Results

The simulation was rigorously tested both with each critter type individually and in collective scenarios. Key observations are as follows:

- **Bears**: Successfully alternated between / and \ as expected. Demonstrated proper wall-following behavior.

- **Tigers**: Correctly changed colors every three moves and dynamically adjusted movements to avoid walls and other critters.

- **White Tigers**: Accurately transitioned from tgr to TGR upon infection while otherwise mirroring the behavior of Tigers.

- **Giants**: Cycled through the text representations fee, fie, foe, and fum at fixed intervals. Exhibited consistent clockwise wall-hugging behavior.

- **NinjaCats**: Dynamically adjusted both color and string representation. Displayed unique and varied movement patterns, distinguishing them from other critter types.

DISCUSSION

Strengths

- Effectively utilized inheritance to simplify the process of adding new critter types.
- Modular class design ensured maintainability and facilitated future extensibility.
- The simulation provided real-time feedback, enabling efficient observation and analysis of critter behaviors.

Limitations

- Behavior logic was closely tied to specific requirements, reducing flexibility for implementing new features.
- Randomization introduced slight inconsistencies in Tiger color patterns during shorter simulation runs.

Future Improvements

- Enable user-defined critter classes to enhance customization and user engagement.
- Optimize the framework to support larger grid sizes and accommodate higher critter counts efficiently.

CONCLUSION

This project successfully demonstrates the application of object-oriented programming principles in the development of a dynamic and interactive simulation. By employing inheritance and polymorphism, the system facilitated the seamless integration of distinct critter behaviors while ensuring code reusability and scalability. The modular framework promoted a clear separation between behavior logic and rendering, simplifying debugging and paving the way for future enhancements.

Randomized elements, such as the Tigers' color changes and the Giants' text cycles, introduced an element of unpredictability, enhancing the overall richness of the simulation. Each critter type functioned as designed, reflecting careful planning and strict adherence to the project requirements.

The project's success underscores the versatility of object-oriented principles in creating flexible, maintainable, and scalable systems, serving as a valuable foundation for future advancements in simulation design.

REFERENCES

- Project specification document.
- Java API documentation for Color and Random classes.
- Course materials on object-oriented programming concepts.

GITHUB LINK

https://github.com/kugendran95/ITWM5043-Software-Development_FinalProject