Optimizing Narrative Choices in Mystic Messenger with Linear Equation Systems

Bertha Soliany Frandi 13523026[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523026@std.stei.itb.ac.id, [2]bertha.soliany@gmail.com

***Abstract*—This paper explores the application of linear equation systems to optimize branching storylines in interactive narrative games, using Mystic Messenger as a case study. As narrative complexity grows, traditional methods like decision trees and flag systems to manage branching paths can lead to inefficiencies and inconsistencies. By using a system of linear equations, developers can systematically model and analyze the dependencies between player choices and outcomes, ensuring coherence and efficiency in narrative design. The study highlights the potential of mathematical frameworks in advancing the future of interactive storytelling.**

***Keywords*—Otome Games, Branching Storylines, System of Linear Equations, Narrative Optimization, Mystic Messenger.**

I. Introduction

Otome games are video games that originate from Japan. This type of game is story-driven games that target female audiences. The word otome itself means maiden in Japanese. The focus of this game is building a romantic relationship with the love interest through interactive storytelling. Because of this interactiveness, almost every choice that the player makes influences the narrative's direction and outcomes. One of the most popular otome games is Mystic Messenger.

Mystic Messenger is an otome game from Cheritz, a game developer from South Korea that was founded in 2012. One of the reasons behind the popularity of this game is the concept of it. In Mystic Messenger, players will mostly interact with fictional characters through a chat app like LINE or WhatsApp. There is also a call feature and an email feature. Following it, Mystic Messenger uses the real-world time to determine when texts and calls come in. So basically, this game stands out due to its real-time chat-based mechanics complemented with an intricate branching storyline that immerses the players in a simulation of texting and phone calls with fictional characters.

All otome games have branching narratives. How complicated the branching is depends on the designer. It is usually presented with multiple choices that shape the story's progression. The player choices will determine the ending of the story. Typically, developers design branching stories using flowcharts or decision trees. There are also developers that only use flags to make a branching storyline. Each choice a player makes determines the subsequent path, creating a web of interconnected storylines. However, as the complexity of the branching narrative grows, maintaining coherence and balancing all possible outcomes becomes increasingly challenging.

One of the challenges in creating complex storylines is ensuring that every branch remains meaningful and engaging. Developers must carefully manage the dependencies between choices to avoid inconsistencies and plot holes. Furthermore, designing and testing every possible outcome can be resource-intensive, making it difficult to optimize the narrative structure effectively. In games like Mystic Messenger that are real-time-based and have a complex branching narrative, it is necessary to have a systematic optimization.

Systems of linear equations offer a novel approach to handling these challenges. By representing choices and the outcomes with mathematical equations, it becomes possible to analyze and optimize the branching structure systematically. The system of linear equations also allows developers to identify redundancies, ensure consistency, and balance the narrative more efficiently. When applied to a game like Mystic Messenger, this approach could enhance the game's branching storylines, ensuring a smoother and more satisfying player experience while reducing development time and complexity.

II. Theoretical Foundations

A. Otome Games and Storyline

As an interactive medium, otome games rely heavily on player interaction to create immersive experiences. Player choices are the driving force behind branching storylines, where each decision leads to distinct paths and potentially different endings. Conventional methods for crafting these narratives include developing decision trees or flowcharts that outline every possible path. There is also a method that uses a flag to determine the storyline outcome. Although this approach works well for simpler games, it becomes unwieldy as the narrative's complexity increases.

Decision trees and flowcharts are used as visual tools to represent all possible narrative paths. In the decision tree, each node represents a choice, and the branches represent

possible outcomes such as further choices or events. Flowcharts provide a similar representation that offers a roadmap of how different decisions interconnect and lead to various endings. Although this method makes it easier to visualize the storyline, it is not the same as implementing it in the development process. As the narrative complexity increases, these tools become less practical due to the exponential growth in the number of branches and interdependencies.

The same goes for using flags. Flags are variables that track specific choices or events made by the player. One of the popular visual novel engines is Ren'Py. As a visual novel engine, Ren'Py offers an easy-to-learn scripting language that efficiently writes large visual novels. At Ren'Py, developers set flags when the player makes decisions and check these flags later to determine the ending. For example, a choice like "Go to the market with character A" could set a flag, and this flag, during a key narrative event, will be checked to determine the outcome.

Flags provide an easy method for managing branches by keeping track of important choices. Nevertheless, as the quantity of flags grows, monitoring their interactions becomes increasingly complicated. For instance, various flag combinations might result in different outcomes, and confirming that these combinations correspond with the desired narrative structure demands careful planning and testing.

B. Mystic Messenger

Fig. 2.1 Mystic Messenger (Source:[4]).

Mystic Messenger is a visual novel game from South Korea, created by Cheritz. It launched on July 8, 2016, and rapidly became popular due to its creative storytelling and interactive features. The game combines conventional visual novel elements with real-time messaging dynamics, offering players a one-of-a-kind experience. The game narrative unfolds through chatrooms and messaging, route-based narrative, and real-time engagement.

The chatrooms, text messages, and phone calls are where the player can interact with the characters. The main story consists of 11 days where at the end of day 4, the player can obtain a character-specific "routes". This route is determined by the amount of "heart" the player has. There are eight hearts with different colors that represent each character. The highest number of hearts will be the player route.

There are seven character-specific routes that the player can obtain. There is Yoosung, Jaehee, Zen, Jumin, 707, V, and Ray. Note that the game has three modes and only a certain route can be obtained depending on the mode the player chooses. The three modes are Casual Story, Deep Story, and Another Story. In Casual Story mode the player can obtain either Yoosung, Jaehee, or Zen route. In Deep Story mode the player can obtain Jumin or 707 route and in the Another Story mode the player can only obtain V or Ray route.

Each route will have seven endings except the after ending and outside of the route ending there is a general ending too. The seven endings are Good Ending, Normal Ending, three Bad Ending, and two Bad Relationship Ending. The general endings are the Bad Ending for each mode (Casual Story, Deep Story, and Another Story). There is a Secret Ending and another ending that can be obtained through DLC. Each of these endings have a requirement that the player must finish. For example, the Bad Ending for story mode can be obtained if the player's highest amount of heart is not the chosen story mode obtainable character route. Another example is to obtain a Good Ending on day 11 the player must successfully invite 10 or above guests and obtain a specific chatroom.

	Zen	Yoosung	Jaehee	Jumin	707	V	Ray
Prologue	Casual & Deep Story Pologue Bad Ending					Another Story Prologue BE	
Day 4						Another Story Bad Ending	
Day 5	Casual Story Bad Ending			Deep Story Bad Ending			
Day 6			BE1 & BRE1				
Day 7	BE1 & BRE1	BE1 & BRE1		BE1 & BRE1	BE1 & BRE1	BE1 & BRE1	BE1 & BRE1
Day 8		BE2	BE2	BE2			
Day 9	BE2				BE2	BE2 & BRE2	BE2
Day 10	BE3 & BRE2	BE3 & BRE2	BE3 & BRE2	BE3 & BRE2	BE3 & BRE2	BE3	BE3 & BRE2
Day 11	Good & Normal Ending	Good & Normal Ending	Good & Normal Ending	Good & Normal Ending	Good & Normal Ending	Good & Normal Ending	Good & Normal Ending
Note: Only endings between day 6-11 are counted in / Endings of each character				BE = Bad Story Ending BRE = Bad Relationship Ending			

Fig. 2.2 All Endings of Mystic Messeng(Source:[4]).

C. Branching stories challenges

There are a couple of challenges for creating a branching storyline. The first one is complexity management. The number of possible outcomes grows exponentially with the addition of choices. This will make it difficult to visualize and manage the overall structure of the storyline. The second one is consistency. Developers must ensure that all branches align with the story and not contradict each other which will make the development process require a lot of resources and time. The third one is balancing player expectations and story quality. Developers must offer significant options while preserving the quality of the story demands thoughtful preparation and repeated refinement.

D. System of linear equations

A system of linear equations is a mathematical idea that includes two or more linear equations that utilize the same variables. These equations together describe the relationships among variables, usually representing lines, planes, or surfaces in higher dimensions within a space. The solution to this system is the group of variable values that satisfies all the equations at the same time, generally corresponding to the points where the equations intersect or share commonality. Systems of linear equations are extensively utilized in mathematics, engineering, and computer science to represent and address issues that involve connections among various variables. Examples of linear equations are $2x_1 + x_2 + 10x_3 = 25$. Systems of linear equations can be represented in several ways. The example before is the standard form of linear equations.

There is matrix form and augmented matrix form that come from $Ax = b$ as shown below.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

$$A \qquad \mathbf{x} \qquad \mathbf{b}$$

Fig. 2.3 Matrix Form of Linear Equation System (Source:[9]).

$$[A \mid \mathbf{b}] = \left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{array}\right]$$

Fig. 2.4 Augmented Matrix Form of Linear Equation System (Source:[9]).

The solution of a system of linear equations can be obtained by using several methods. There is a substitution method, elimination method, graphical method, and matrix method. Substitution method is done by substituting a variable with the known value. The Elimination method is done by adding or subtracting equations to eliminate variables to reduce the system to fewer equations. The graphical method is done by making a graph with the equation representing where the solutions are the intersections of lines or hyperplanes. Matrix form of linear equations can be solved using Gaussian elimination, Gaussian-Jordan method, inverse matrix method, LU decomposition, and many more. One of the simplest ways to solve a matrix equation is using Gaussian elimination where the augmented matrix form is transformed into a row echelon form and solved using back substitution.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} & b_n \end{bmatrix} \sim \text{OBE} \sim \begin{bmatrix} 1 & * & * & \dots & * & * \\ 0 & 1 & * & \dots & * & * \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \vdots & 1 & * \end{bmatrix}$$

Fig. 2.5 Gaussian Elimination (Source:[9]).

Talking about solutions of linear equations, there are three solution types that a system can have. There is a unique solution, infinite solution, and no solution. One of the ways to determine the types of solutions is by using Gaussian elimination and see the row echelon form of it. If the matrix has one or more rows that all its values are zero, the system has an infinite number of solutions. If there is a row where all its values are zero except the last column, the system does not have a solution. Except the matrix did not satisfy the two conditions before, the system has a unique solution.

For the cases of storyline, the variables represent choices and the equations represent outcomes. With this representation it becomes possible to identify dependencies between choices and outcomes as the program systematically analyzes the entire narrative structure, detect redundant or conflicting narrative paths, and ensure the branching structure aligns with the intended narrative design so it optimizes the branching paths for consistency and efficiency.

III. IMPLEMENTATION

The implementation of linear equation systems in optimizing narrative choices involves several steps. First, the requirements for each ending must be researched and planned out. Second, determining the amount of matrix that is needed for the storyline. In this step, the variable that the system of linear equations has must be determined. Each of the variables represents different requirements. The decision to use a matrix compared to other forms is due to solving the equation. Using a matrix enables the use of computational tools, where in this case NumPy is used.

The implementation is done by using Python and the NumPy library. The core functionalities include character route selection, evaluation of choices through linear equations, and determining endings. Besides that, there are a series of complement functions and dialogue to make an experience like the original game.

A. Story Mode and Prologue

At the start of the program, the user will be greeted and asked to choose a story mode. Story mode will determine which character route the user can get. There are Casual Story, Deep Story, and Another Story. After selecting a story mode, there will be a prologue story like the original prologue story in the Mystic Messenger game. If the user continues to answer the question with “I’m calling the police,” a prologue bad ending will be triggered. The implementation of this bad ending did not use a system of linear equations because of the simple requirement it has.

```
prologue_choices = []

print("\nPrologue")
print("Note that there's a bad ending if all choices are the same (specific choice).")
print("\nThere's suddenly a strange app on your phone. It is a chatroom app. You decide to open it.")

prologue_dialogues = [
    "You see a message pop up on the app. It says to do crime. What will you do?",
    "This app feels strange... Are you sure about your choice?",
    "The chatroom becomes more intense. Is this your final answer?",
    "Are you sure you want to continue? The app is getting weirder."
]

for i in range(4):
    time.sleep(1)
    print("\n"+prologue_dialogues[i])
    print("1. GLADLY!")
    print("2. I'm sorry, I can't do that")
    print("3. Prf, pay me first")
    print("4. I'm calling the police")
    try:
        choice = int(input("Enter your choice (1/2/3/4): "))
        if choice not in [1, 2, 3, 4]:
            print("Invalid choice. Please try again.")
            i -= 1
            continue
        prologue_choices.append(choice)
    except ValueError:
        print("Invalid input. Please enter a number between 1 and 4.")
        i -= 1

if prologue_choices == [4,4,4,4]:
    time.sleep(1)
    print("\nThe screen flickers. You feel like something's watching you.")
    print("\nBad Ending: Just don't call the police...")
    exit()
else:
    time.sleep(1)
    print("\nWhatever your choices, atleast you're not insisting to call the police 4 times. Let's move on.")
```

Fig. 3.1 Prologue Implementation (Source: Author).

The code asks the user for four inputs and checks if the user input is constantly number 4. If yes, the bad ending will be triggered. This portrays the interaction between the player and Unknown in the original game, where a bad ending will happen if the player constantly chooses a dialogue that shows a lie and causes Unknown to be annoyed.

B. Character Route

After the prologue, the user will be brought to adding a heart to characters. This is the implementation of the colored hearts that each of the characters have. It symbolizes affinity to the user. The character with the highest hearts will determine the route. Not only the number of hearts, but the story mode that the user chose at

the start of the program will also determine whether a bad ending will be triggered or not.

In the original game, hearts are obtained by interacting with the characters through chatrooms and phone calls. But in this program, to simplify that process, the user can directly set the amount of each character heart.

```python
def determine_story_route(story_mode, characters):
    """
    Determine the character route based on story mode and hearts.
    :param story_mode: Selected story mode (Casual, Deep, Another)
    :param characters: List of CharacterRoute objects
    :return: The selected character's route
    """
    # Filter characters by story mode
    eligible_characters = [char for char in characters if char.story_mode == story_mode]

    if not eligible_characters:
        return None  # No characters available for the selected story mode

    # Determine character with highest hearts
    selected_character = max(eligible_characters, key=lambda char: char.hearts)

    return selected_character
```

Fig. 3.2 Function for Determine Story Route (Source: Author).

The code above is a function to determine which route the user obtains. This code checks which character has the highest number of hearts. If the mentioned character is a character that is available in the chosen story mode (is in *eligible_characters*), the user gets that character route. If the mentioned character is not a character that can be obtained in the chosen story mode, the user won't be able to continue because there is no route, and the program will exit.

There is also a bad ending that can be triggered here. Following the determination of character route, the *mode_bad_ending* function will be called. This function will return True if the character with the highest number of hearts is not an eligible character for the corresponding story mode.

```python
def mode_bad_ending(story_mode, characters):
    """
    Check if Story Mode Bad Ending should be triggered.
    :param story_mode: Selected story mode
    :param characters: List of CharacterRoute objects
    :return: True if Story Mode Bad Ending should be triggered, False otherwise
    """
    # Find character with highest hearts
    character_with_highest_hearts = max(characters, key=lambda char: char.hearts)

    # Check if the character with highest hearts is eligible for the selected story mode
    if character_with_highest_hearts.story_mode != story_mode:
        return True
    return False

if mode_bad_ending(selected_story_mode, characters):
    if selected_story_mode == "Casual":
        print("\nBad Ending triggered! You must have either Jaehee's, Zen's, or Yoosung's heart highest.")
    elif selected_story_mode == "Deep":
        print("\nBad Ending triggered! You must have either 707's or Jumin's heart highest.")
    elif selected_story_mode == "Another":
        print("\nBad Ending triggered! You must have either V's or Ray's heart highest.")
    else:
        print("\nBad Ending triggered! No character route available.")
    exit()
```

Fig. 3.3 Function for Story Mode Bad Ending (Source: Author).

C. Choices

The gameplay of Mystic Messenger involves making choices. This choice will determine the ending that the player got. For this program, predetermined choices are given. There is one for each ending.

```python
print("\nPredefined choices:")
if selected_story_mode == "Casual" or selected_story_mode == "Deep":
    print("1. For Good Ending: 10, 1, 10")
    print("2. For Normal Ending: 10, 1, 9")
    print("3. For Bad Relationship Ending: 0.2, 1, 1")
    print("4. For Bad Ending: 1, 1, 1")
    choice_input = input("Enter your choice (1/2/3/4): ")
    if choice_input == "1":
        player_choices = [10, 1, 10]
        break
    elif choice_input == "2":
        player_choices = [10, 1, 9]
        break
    elif choice_input == "3":
        player_choices = [0.2, 1, 1]
        break
    elif choice_input == "4":
        player_choices = [1, 1, 1]
        break
    else:
        print("Invalid choice. Please try again.")
```

Fig. 3.4 Choices Implementation (Source: Author).

Variable *player_choices* is the predetermined choice. It stores an array of integers. It represents a requirement that the player has or obtained. The array of integers can be seen as x_1, x_2, and x_3. For good and normal ending, x_1 represents the number of hearts, x_2 represent whether the player interacts in a specific chatroom, and x_3 represent the number of guests that attend the party in day 11. For bad relationship ending, x_1 represents a participation rate in chatrooms, x_2 represent whether player interact in a specific chatroom, and x_3 represents whether the player follows a specific story. For a bad ending, x_1 represents if the player chooses a specific dialogue, x_2 represents whether the player interacts in a specific chatroom, and x_3 represents whether the player follows a specific story.

```python
# Initialize data for story mode
universal_matrix = np.array([
    [1, 1, 1],   # Bad Ending
    [0.3, 1, 1], # Bad Relationship Ending
    [1, 1, 1],   # Good Ending
    [1, 1, 1]    # Normal Ending
])

thresholds_casual = [3, 2.3, 12, 9]  # Casual Story mode
thresholds_deep = [3, 2.3, 12, 9]    # Deep Story mode
thresholds_another = [3, 2.3, 17, 16]  # Another Story mode
```

Fig. 3.5 Matrix form of System of Linear Equation (Source: Author).

This variable will be calculated with the *universal_matrix* that has been set before. The *universal_matrix* represents a system of linear equations. It is used to calculate contributions to different endings based on the predefined choices. It is applied via a dot product operation with *player_choices* to produce results which are then compared against thresholds to determine ending.

On the first row is the coefficients to the bad ending, the second row is the bad relationship ending coefficients, the third row is the coefficients for a good ending, and the last row is for the normal ending. Each of the coefficients is related to the x_1, x_2, and x_3. that the *player_choices* represent.

D. Ending

The last core functionality is how the ending is determined. *determine_ending* function is the one to do that. *player_choices*, *universal_matrix*, and thresholds, the ending can be determined.

```
results = np.dot(self.matrix, choices)
```

Fig. 3.6 Equation for Results Using Dot Product (Source: Author).

This code produces a result array where each element represents the calculated contribution for a specific ending. $results[0]$ for the bad ending, $results[1]$ for the bad relationship ending, $results[2]$ for the good ending, and $results[3]$ for the normal ending. Let us take $[0.2, 1, 1]$ for the example of *player_choices*. $[0.2, 1, 1]$ will be calculated with *universal_matrix* resulting in this calculation:

$$results[0] = (1 \times 0.2) + (1 \times 1) + (1 *\times 1) = 0.2 + 1 + 1 = 2.2$$
$$results[1] = (0.3 * 0.2) + (1 * 1) + (1 * 1) = 0.06 + 1 + 1 = 2.06$$
$$results[2] = (1 * 0.2) + (1 * 1) + (1 * 1) = 0.2 + 1 + 1 = 2.2$$
$$results[3] = (1 * 0.2) + (1 * 1) + (1 * 1) = 0.2 + 1 + 1 = 2.2$$
$$results = [2.2, 2.06, 2.2, 2.2]$$

Using the result of the dot product, each value of results is compared against corresponding thresholds to determine if an ending is achievable. For example, the condition to bad ending is checking $thresholds[0]$. $results[0]$ value is 2.2 and $thresholds[0]$ value is 3. $results[0]$ is less than $thresholds[0]$ so the bad ending is not achieved.

```python
def determine_ending(self, choices, story_mode):
    """
    Determine the player's outcome for this character's route based on choices and story mode.
    :param choices: List of player choices [x1, x2, x3].
    :param story_mode: Selected story mode (Casual, Deep, Another).
    :return: Outcome string for the character's route.
    """
    if self.is_locked:
        return f"Route locked. {self.name}'s route already achieved."

    # Hitung dot product
    results = np.dot(self.matrix, choices)

    if story_mode == "Casual":
        if results[2] >= self.thresholds[2] and choices[2] >= 10:
            return f"Good Ending: Congratulations! You have become lover with {self.name}"
        if results[3] >= self.thresholds[3] and choices[2] <= 9:
            return f"Normal Ending: You have a good time with {self.name}"
        if results[1] >= self.thresholds[1] and choices[0] < 0.3:
            return f"Bad Relationship Ending: You disappoint {self.name}"
        if results[0] >= self.thresholds[0] and choices[0] == 1 and choices[1] == 1 and choices[2] == 1:
            return "Bad Ending: Why aren't you try harder?"

    elif story_mode == "Deep":
        if results[2] >= self.thresholds[2] and choices[2] >= 10:
            return f"Good Ending: Congratulations! You have become lover with {self.name}"
        if results[3] >= self.thresholds[3] and choices[2] <= 9:
            return f"Normal Ending: You have a good time with {self.name}"
        if results[1] >= self.thresholds[1] and choices[0] < 0.3:
            return f"Bad Relationship Ending: You disappoint {self.name}"
        if results[0] >= self.thresholds[0] and choices[0] == 1 and choices[1] == 1 and choices[2] == 1:
            return "Bad Ending: Why aren't you try harder?"

    elif story_mode == "Another":
        if results[2] >= self.thresholds[2] and choices[2] >= 17:
            return f"Good Ending: Congratulations! You have become lover with {self.name}"
        if results[3] >= self.thresholds[3] and choices[2] <= 16:
            return f"Normal Ending: You have a good time with {self.name}"
        if results[1] >= self.thresholds[1] and choices[0] < 0.3:
            return f"Bad Relationship Ending: You disappoint {self.name}"
        if results[0] >= self.thresholds[0] and choices[0] == 1 and choices[1] == 1 and choices[2] == 1:
            return "Bad Ending: Why aren't you try harder?"

    return "No Ending Achieved"
```

Fig. 3.7 Function for Determining Ending (Source: Author).

IV. Conclusion

This study demonstrates how a system of linear equations can be leveraged to optimize branching storylines in interactive otome games like Mystic Messenger. Traditional methods, such as decision trees and flags, while effective for simple narratives, often become cumbersome as the complexity of the storyline increases. By adopting a system of linear equations, developers can efficiently model, analyze, and refine narrative structures to ensure consistency and player satisfaction.

The integration of linear equation systems provides a scalable solution to manage dependencies between player choices and outcomes, offering greater flexibility and coherence in narrative design. This approach not only reduces development time but also enhances the player's experience by delivering well-structured and engaging storylines. As gaming continues to evolve, the use of mathematical frameworks in narrative optimization presents exciting opportunities for the future of interactive storytelling.

V. Appendix

The Github repository for source code can be accessed at https://github.com/BerthaSoliany/makalah-algeo.

VI. Acknowledgment

The author would like to express gratitude to God for providing strength and clarity, to the game development community for their shared knowledge, and to Mr. Rinaldi Munir for his guidance in Linear Algebra and Geometry. The author is also grateful to Cheritz, the developers of Mystic Messenger, for creating a groundbreaking game that inspired this study. Lastly, the author extends gratitude to the family for the support and encouragement throughout this study.

References

[1] Shunsuke Mukae, "30 years of otome game: How did the term 'otome game' spread (Online source)," ResearchGate. [Online]. Available: https://www.researchgate.net/publication/379567783_30_years_of_otome_game-No1How_did_the_term_'otome_game'_spread. [Accessed: Jan. 1, 2025].

[2] KoreaScience, "Analysis of otome game trends (Online source)," KoreaScience. [Online]. Available: http://koreascience.or.kr/article/JAKO202428443273575.page. [Accessed: Jan. 1, 2025].

[3] VICE, "Mystic Messenger Review (Online source)," VICE. [Online]. Available: https://www.vice.com/en/article/mystic-messenger-review/. [Accessed: Jan. 1, 2025].

[4] Mystic Messenger Wiki, "Mystic Messenger (Online source)," Mystic Messenger Wiki. [Online]. Available: https://mystic-messenger.fandom.com/wiki/Mystic_Messenger_Wiki. [Accessed: Jan. 1, 2025].

[5] RenPy, "Ren'Py Visual Novel Engine (Online source)," RenPy. [Online]. Available: https://www.renpy.org/. [Accessed: Jan. 1, 2025].

[6] FasterCapital, "Interactive storytelling: The art of story branching (Online source)," FasterCapital. [Online]. Available: https://fastercapital.com/content/Interactive-storytelling--Story-Branching--The-Art-of-Story-Branching-in-Interactive-Media.html?utm_source=chatgpt.com. [Accessed: Jan. 1, 2025].

[7] AAAI, "Agency play: Dimensions of agency for interactive narrative design (Online source)," AAAI. [Online]. Available: https://aaai.org/papers/0008-ss09-06-008-agency-play-dimensions-of-agency-for-interactive-narrative-design/?utm_source=chatgpt.com. [Accessed: Jan. 1, 2025].

[8] GeeksForGeeks, "System of linear equations (Online source)," GeeksForGeeks. [Online]. Available: https://www.geeksforgeeks.org/system-linear-equations/. [Accessed: Jan. 1, 2025].

[9] R. Munir, "Aljabar Geometri: Sistem Persamaan Linier," IF2123 Aljabar Linear dan Geometri, 2023. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-03-Sistem-Persamaan-Linier-2023.pdf. [Accessed: Jan. 1, 2025].

[10] R. Munir, "Tiga Kemungkinan Solusi SPL," IF2123 Aljabar Linear dan Geometri, 2023. [Online]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://informatika.

stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-04-Tiga-Kemungkinan-Solusi-SPL-2023.pdf. [Accessed: Jan. 1, 2025].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Januari 2025

Bertha Soliany Frandi
13523026