Analysis of Algorithm Complexity in the Arcade Rhythm Game Maimai DX

Julian Caleb Simandjuntak (13522099)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522099@std.stei.itb.ac.id

***Abstract*—Maimai DX is an arcade rhythm game that is famous among young people, especially in Asia. There are various types of songs and difficulty levels that make players addicted. One of the most interesting things about maimai DX is its scoring system, where each song chart can reach not 100% but 101% or All Perfect Plus, making it even more challenging for players to complete a chart perfectly. In this paper, we will analyze the complexity of the algorithm for the scoring system in maimai DX and determine the complexity of the algorithm using Big-O Notation.**

***Keywords*—Algorithm Complexity, Time Complexity, Big-O Notation, input size**

I. Introduction

Maimai, a captivating arcade rhythm game developed by Sega, has gained popularity not only in its home country of Japan but also in various other countries, including Indonesia. This game is renowned for its international appeal, with versions available in multiple languages, including Japanese, English (International Version), and Simplified Chinese (Chinese Version).

The gameplay in maimai revolves around rhythmically pressing eight buttons surrounding the screen or interacting directly with the screen itself. Players synchronize their actions with the rhythm of the chosen song, responding to symbols in the form of rings or slides that emanate from the center of the screen. Whether you prefer to tackle the challenges solo or enjoy the competitive spirit with friends, maimai accommodates both single-player and multiplayer modes, supporting up to four players.

Over the years, maimai has evolved, giving rise to several series, such as maimai, maimai PLUS, maimai MiLK, maimai Finale, and the focus of this paper, maimai DX. This second-generation installment has further branched into various subseries, such as maimai DX, maimai DX PLUS, maimai DX Splash, maimai DX Festival, and the recently released maimai DX Buddies in September. These iterations offer a diverse array of songs, encompassing Niconico and Vocaloid tunes, the iconic Touhou Project melodies, anime tracks, and even original compositions.

While the scoring systems across the maimai series share similarities, they also incorporate unique elements. In this paper, our primary focus will be on maimai DX due to its particularly intricate and compelling scoring system, providing an ideal basis for a comprehensive analysis of algorithm complexity.

II. Basic Theory

Algorithm complexity is a fundamental concept in computer science that helps us analyze and compare the efficiency of different algorithms when solving a specific problem. There are 2 types of algorithm complexity, namely time complexity (based on the number of computational stages as a function of the number of inputs n, denoted as T(n)) and space complexity (based on the memory used by the data structure, denoted as S(n)). The complexity of this algorithm is then expressed in O(n), or known as Big-O notation. The Big-O notation is a key tool used to describe the upper bound on the time or space complexity of an algorithm in the worst-case scenario. It provides a concise and abstract way to represent how the performance of an algorithm scales as the size of the input data increases. In the Big-O notation, we typically use the letter "O" followed by a function that represents the upper bound, denoting the order of growth. For example, O(n) signifies linear time complexity, where the algorithm's execution time grows proportionally to the input size.

Let's consider an example to illustrate how different algorithms can be used to solve the same problem and how their complexities can vary. Suppose we want to find the maximum element in an unsorted list of numbers. Two common algorithms for this task are the "linear search" and the "divide and conquer" approach. In a linear search (2), the algorithm examines each element one by one, making it O(n) in terms of time complexity because the execution time increases linearly with the input size.

```
KAMUS GLOBAL
constant CAPACITY : integer = 100
constant IDX_UNDEF : integer = -1
constant MARK : integer = -9999

type ElType : integer { elemen list }
type List : < contents: array [0..CAPACITY-1] of ElType {penyimpanan elemen List. },
              nEff: integer ≥ 0 { jumlah elemen efektif List. } >

function length(l: List) → integer
        { Prekondisi : l terdefinisi.
          Mengirimkan banyaknya elemen efektif l, 0 jika list kosong. }
```

(1)

```
function linearSearch(l: List, x: ElType) → integer
    { Prekondisi : l, x terdefinisi.
    Mengembalikan indeks elemen pertama l yang bernilai x (jika ada),
    atau mengembalikan IDX_UNDEF jika tidak ada. }

    KAMUS LOKAL
    i : integer
    found : boolean

    ALGORITMA
    found <- false
    i <- 0
    while (i < length(l) and not(found)) do
        if (l.contents[i] = x) then
            found <- true
        else
            i <- i + 1
    { i = length(l) or found }
    if (found) then
        -> i
    else
        -> IDX_UNDEF
```

(2)

On the other hand, a divide and conquer algorithm, such as binary search (3) when the list is sorted, has a time complexity of O(log n), which is significantly more efficient as it divides the problem into smaller subproblems and can find the maximum element faster, especially for larger datasets. By analyzing and comparing the Big-O notations of these two algorithms, we can make an informed decision about which one to use based on the problem's size and constraints, ensuring we select the most efficient solution. This fundamental understanding of algorithm complexity is crucial for designing efficient and scalable software systems.

```
function binarySearch(l: List, x: ElType) → integer
    { Prekondisi : l terurut membesar, x terdefinisi.
    Mengembalikan indeks elemen pertama l yang bernilai x
    (jika ada), atau mengembalikan IDX_UNDEF jika tidak ada. }

    KAMUS LOKAL
    low, mid, high : integer
    found : boolean

    ALGORITMA
    found <- false
    low <- 0
    high <- length(l) - 1
    while (low <= high and not(found)) do
        mid <- (low + high) / 2
        if (l.contents[mid] = x) then
            found <- true
        else
            if (l.contents[mid] < x) then
                low <- mid + 1
            else
                high <- mid - 1
    { low > high or found }
    if (found) then
        -> mid
    else
        -> IDX_UNDEF
```

(3)

III. Maimai DX Scoring System

Before we start analyzing the algorithm, we will try to look at the scoring system behind maimai itself [1]. It will be divided into 2, for each song chart and the rating increase for each player..

A. The Song Chart

Maimai is a rhythm game where players have to press buttons or screens according to the timing of the song which is visualized in the form of a song chart containing various types of notes. Notes on maimai are divided into 4, namely tap (regular press, the note moves from the inside to the edge), hold (long press, the note moves from the inside to the edge), slide (press and drag, the note appears, usually from the edge and has a route which also ends at the edge), touch (a silent tap on the screen), touch hold (a silent hold on the screen), paired (notes that come out at the same time), break (a special note that has added value) and EX note (an auto perfect note). Each note that is pressed will be assessed for timing, and the game will issue a judgment called "Judgements", which is divided into 5, namely Miss, Good, Great, Perfect, and Critical Perfect. In addition, maimai will also tell you whether the note was pressed too fast (Early) or too late (Late).

Maimai measures the score obtained by players on the chart as "Achievement". What makes maimai unique is that the achievement can be achieved from 0 to a maximum of 101%, with 100% obtained by pressing all notes except the break with Perfect or Critical Perfect, and 1% of the break bonus which is different for each judgment. Player have to score 80% or higher to successfully pass the chart.

The following are the points of each note with each judgment,

Table 1 Points of each Note and Judgement

Note	Good	Great			Perfect		Critical
		Low	Mid	High	Low	High	
Tap	250	400	400	400	500	500	500
Hold	500	800	800	1000	1000	1000	1000
Slide	750	1200	1200	1500	1500	1500	1500
Break base	1000	1250	1500	2000	2500	2500	2500
Break bonus	30	40	40	40	50	75	100

In the table above, note tap has the same value as note touch and note hold has the same value as note touch hold. Breaks have a special assessment with break base for base achievement and break bonus for maximum achievement.

Base achievement is obtained by adding up all points obtained except break bonuses, then dividing it by the maximum total base points and multiplying it by 100. For achievement bonuses, adding up all break bonus points and dividing it by the maximum total break bonus points, without multiplying by 100. Total achievement obtained by adding up the base achievement and bonus achievement.

For example, suppose in a chart play:

- Out of 50 tap notes, 1 miss and 49 perfect
- Of the 10 break tap notes, 1 great high, 1 perfect low, 2 perfect high, and 6 critical perfect
- Of the 4 slide notes, 4 are perfect
- Of the 2 break slide notes, 2 are critical perfect
- Of the 9 hold notes, 1 is great and 8 are perfect

The following is the base achievement calculation:

- Tap notes = 49 * 500 = 24500
- Break tap notes = 1 * 2000 + 1 * 2500 + 2 * 2500 + 6 * 2500 = 24500
- Slide notes = 4 * 1500 = 6000

- Break slide notes = 2 * 2500 = 5000
- Hold notes = 1 * 800 + 8 * 1000 = 8800
- Total = 68800 points

The following is the achievement bonus calculation:

- Break tap notes = 1 * 40 + 1 * 50 + 2 * 75 + 6 * 100 = 840
- Break slide notes = 2 * 100 = 200
- Total = 1040 points

The following is the calculation of max base achievement:

- Tap notes = 50 * 500 = 25000
- Break tap notes = 10 * 2500 = 25000
- Slide notes = 4 * 1500 = 6000
- Break slide notes = 2 * 2500 = 5000
- Hold notes = 9 * 1000 = 9000
- Total = 70000 points

The following is the calculation of the max achievement bonus

- Break tap notes = 10 * 100 = 1000
- Break slide notes = 2 * 100 = 200
- Total = 1200 points

Based on the calculation above, the player's total achievement is (68800/70000) * 100 + (1040/1200) = 99.1523 (in percent).

B. The Rating System

Maimai DX has a rating feature as an approximation of how high a player's abilities are based on the number of charts played and the achievements obtained. The formula for calculating the rating obtained from a song chart is

DX rating points = Chart constant * Achievement * Score multiplier

(4)

There are 5 types of song chart difficulty in maimai, from easiest to hardest, Basic, Advanced, Expert, Master, and Re:Master (not all song has Re:Master difficulty). Each difficulty is accompanied by a value or level that indicates how difficult the chart is in general compared to other charts. For example, the Oshama Scramble Master difficulty in the thumbnail shows the number 13, which means the chart is level 13. Chart constant is a hidden value that better describes the level of difficulty of a chart. Oshama Scramble has a chart constant of 13.6. Ordinary charts usually have a chart constant between 0 and 0.6, but for charts with a plus (+) level such as 13+, they have a chart constant of more than equal to 0.7.

Score multipliers are multipliers based on the achievements obtained. The maimai DX score multiplier is as follows

Table 2 Maimai DX Score Multiplier

Achievement	Rank	Score multiplier
100.5%	SSS+	0.224
100.4999%	SSS	0.222
100%	SSS	0.216
99.9999%	SS+	0.214
99.5%	SS+	0.211
99%	SS	0.208
98.9999%	S+	0.206
98%	S+	0.203
97%	S	0.2
96.9999%	AAA	0.176
94%	AAA	0.168
90%	AA	0.152
80%	A	0.136
79.9999%	BBB	0.128
75%	BBB	0.120
70%	BB	0.112
60%	B	0.096
50%	C	0.08
40%	D	0.064
30%	D	0.048
20%	D	0.032
10%	D	0.016

Ratings on maimai DX only consider the top plays of the 50 songs played charts, with 15 charts released in the latest version and 35 from the old version. Each time a player sets a new record for a chart, the game calculates the number of points for his game, then performs the following check:

- If the chart being played is new, operate the calculation on the list containing the new chart. If the chart is old, operate on the old chart list.
- If the list already contains playing charts, change the old rating.
- If the list is not full, add it to the list.
- Else find the lower score point chart ranking and replace it with the new one.

A player's DX rating is simply the sum of all individual chart game ratings on the old and new versions of the list.

For example, suppose a player gets 99.8056% on the Yurushite Master chart, which has a chart constant of 14.4, then the DX rating points are 14.4 * 99.8056 * 0.211 = 303.25 or 303 DX rating points.

C. Other Type of Scoring

Apart from achievement, there are 3 other types of scoring on each chart. The first is Full Combo Tiers, which consists of

- Full Combo: Each note gets at least a Good judgment

- Full Combo Plus: Every note gets at least a Great Judgment
- All Perfect: Every note gets at least a Perfect judgment
- All Perfect Plus: Every note gets at least a Perfect judgment and every break note gets a Critical Perfect judgment

Second, there is an achievement rank. Achievement rank is an achievement symbolized in the form of letters. Achievement rank can be seen in table 2.

Lastly, there is the DX score. DX score is a simple metric of player accuracy that doesn't care about modifiers, just focuses on judgment notes. The following are the points for each judgment.

Table 3 DX Point for each Judgement

Judgement	Points
Critical Perfect	3
Perfect	2
Great	1
Good & Miss	0

From the total DX score, the number of stars obtained by the player will be calculated with the following percentage.

Table 4 Number of Stars

Percentage	Number of stars
97%	5
95%	4
93%	3
90%	2
85%	1
< 85%	0

For example, if the maximum DX score is 350, then it is required

- 298 DX score for 1 star
- 315 DX score for 2 stars
- 326 DX score for 3 stars
- 333 DX score for 4 stars
- 340 DX score for 5 stars

Especially for sync games, maimai will give awards based on the performance of both players. The awards consist of Full Sync (Full Combo), Full Sync Plus (Full Combo same difficulty), Full Sync DX (Full Combo Plus same difficulty), and Full Sync DX Plus (All Perfect same difficulty).

IV. Algorithm Complexity

The calculation of the complexity of the maimai DX game scoring algorithm will be divided into 3, providing boundaries to reduce the scope of calculations, making a simple maimai DX game scoring simulation, and calculating the complexity of the algorithm itself based on the simulation created.

A. Boundaries

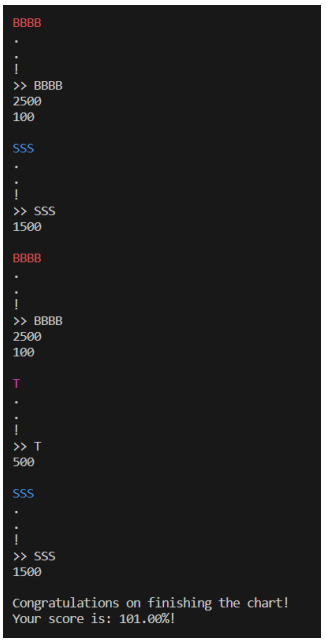
Maimai DX is an arcade game that consists of several parts, starting from player registration/login, character selection, chart settings, song selection, the game chart itself, game chart results, adding ratings, and many more. To simplify and shorten the scope of algorithm complexity analysis, it is carried out by only calculating the point score for each play in a song chart.

B. Simulation

Based on the data on how maimai score points are calculated in the previous chapter, a simple program is created in Python that will simulate a maimai DX song chart game. The program created is not a one-to-one representation of the original maimai DX, but only to show roughly what the scoring system (according to boundaries) is like on maimai DX. The program used in this can be accessed at https://github.com/Julian-Caleb/maimaidx-scoring-system. The program might be further developed in the future

Before decomposing the program, here is how the program will look when run.

(4)



(5)

The program will be decomposed into 3 parts, main function, the gameplay and scoring system itself, and

additional functions which may affect the complexity of the algorithm. First, the main function.

```python
# Main function
if __name__ == "__main__":

    # Buat random chart
    # GenerateRandomList()

    # Ascii Art pembuka
    PrintArt()

    # Username
    name = input("Enter your username: ")
    print(f"Welcome {name}!\n")

    # Sekali permainan maimai DX dapat bermain hingga 4 chart lagu
    for i in range (4) :
        print(f"GAME {i+1} / 4!\n")

        # Memilih lagu
        print("Select a song:")
        print("1. Hibana")
        print("2. Ghost Rule")
        print("3. Daydream Cafe")
        song = int(input(">> "))
        print("")

        # Memilih kesulitan
        print("Select a difficulty:")
        print("1. Basic")
        print("2. Advanced")
        print("3. Expert")
        print("4. Master")
        diff = int(input(">> "))
        print("")
```

(6)

```python
        # Load chart
        chart = LoadNotes(song, diff)

        # Debug chart
        # print(chart)
        # print(LengthChart(chart))
        # print(LengthBreakChart(chart))

        # Play chart
        print("Turn on your capslock!")
        play = input("Ready to play? (Y/N) ")
        print()
        if (play == 'Y') :
            PlayChart(chart)
        else :
            print("You are skipping this song!")

    print("Thank you for playing!")
```

(7)

The main function starts by printing an ascii art followed by asking for a username. Then, in one play of Maimai DX, players can play 4 song charts. Therefore, it loops 4 times to ask for song input and difficulty (assuming the player input is valid). Next, the chart is taken based on the input and asks for input once again whether the player wants to play this chart or just skip it.

The following is the contents of the LoadNotes function and dictionary of charts.

```python
# Database chart lagu
charts = {
    'Hibana': {
        'Basic': ['BBBB', 'SSS', 'BBBB', 'T', 'SSS'],
        'Advanced': ['HH', 'SSS', 'BBBB', 'HH', 'SSS', 'T', 'T', 'SSS'],
        'Expert': ['HH', 'HH', 'HH', 'SSS', 'SSS', 'BBBB', 'HH', 'SSS', 'BBBB', 'SSS'],
        'Master': ['SSS', 'SSS', 'T', 'HH', 'HH', 'SSS', 'BBBB', 'T', 'SSS', 'SSS', 'SSS', 'SSS']
    },
    'Ghost Rule': {
        'Basic': ['SSS', 'T', 'SSS', 'BBBB', 'T'],
        'Advanced': ['SSS', 'SSS', 'SSS', 'BBBB', 'BBBB', 'HH', 'T', 'SSS'],
        'Expert': ['T', 'SSS', 'BBBB', 'HH', 'HH', 'HH', 'SSS', 'BBBB', 'T', 'HH'],
        'Master': ['HH', 'BBBB', 'BBBB', 'HH', 'SSS', 'T', 'T', 'T', 'HH', 'BBBB', 'SSS', 'BBBB']
    },
    'Daydream Cafe': {
        'Basic': ['HH', 'BBBB', 'T', 'SSS', 'BBBB'],
        'Advanced': ['SSS', 'SSS', 'BBBB', 'SSS', 'BBBB', 'HH', 'HH', 'HH'],
        'Expert': ['BBBB', 'T', 'SSS', 'T', 'T', 'SSS', 'T', 'BBBB', 'BBBB', 'SSS'],
        'Master': ['T', 'SSS', 'HH', 'BBBB', 'HH', 'HH', 'T', 'T', 'T', 'SSS', 'BBBB', 'HH']
    }
}
```

(8)

```python
# Load lagu dan difficulty
def LoadNotes(songChoice, diffChoice):
    songList = {1: 'Hibana', 2: 'Ghost Rule', 3: 'Daydream Cafe'}
    diffList = {1: 'Basic', 2: 'Advanced', 3: 'Expert', 4: 'Master'}

    songName = songList[songChoice]
    diffName = diffList[diffChoice]

    return charts[songName][diffName]
```

(9)

The LoadNotes function takes the input number of songs and difficulty, converts them to strings, and matches them against dictionary charts. The function returns an array of strings representing the chart/note sequence of the song. The charts used here (including their lengths) do not correspond to the original maimai DX, but are just assumptions.

Next, here are the contents of the PlayChart function, which is a function for playing charts and evaluating plays.

```python
# Fungsi permainan chart
def PlayChart(noteSequence):
    baseAchievement = 0
    bonusAchievement = 0

    # Mengecek setiap note
    for note in noteSequence:
        print(ColoredOutput(note))
        time.sleep(0.5)
        print(".")
        time.sleep(0.5)
        print(".")
        time.sleep(0.5)
        print("!")

        startTime = time.time()
        userInput = input(">> ")
        endTime = time.time()
        reactionTime = endTime - startTime
```

(10)

```python
# Menghitung Base Point
if userInput == note:
    if 0 < reactionTime <= 0.4:
        print(f"{notes[note]['critical']}")
        baseAchievement += notes[note]['critical']
    elif 0.4 < reactionTime <= 0.7:
        print(f"{notes[note]['perfectHigh']}")
        baseAchievement += notes[note]['perfectHigh']
    elif 0.7 < reactionTime <= 1:
        print(f"{notes[note]['perfectLow']}")
        baseAchievement += notes[note]['perfectLow']
    elif 1 < reactionTime <= 1.2:
        print(f"{notes[note]['greatHigh']}")
        baseAchievement += notes[note]['greatHigh']
    elif 1.2 < reactionTime <= 1.4:
        print(f"{notes[note]['greatMid']}")
        baseAchievement += notes[note]['greatMid']
    elif 1.4 < reactionTime <= 1.5:
        print(f"{notes[note]['greatLow']}")
        baseAchievement += notes[note]['greatLow']
    elif 1.5 < reactionTime <= 1.7:
        print(f"{notes[note]['good']}")
        baseAchievement += notes[note]['good']
    else:
        print("0")
else:
    print("0")
```

(11)

```python
# Menghitung break bonus point
if userInput == 'BBBB':
    if 0 < reactionTime <= 0.4:
        print("100")
        bonusAchievement += 100
    elif 0.4 < reactionTime <= 0.7:
        print("75")
        bonusAchievement += 75
    elif 0.7 < reactionTime <= 1:
        print("50")
        bonusAchievement += 50
    elif 1 < reactionTime <= 1.2:
        print("40")
        bonusAchievement += 40
    elif 1.2 < reactionTime <= 1.4:
        print("40")
        bonusAchievement += 40
    elif 1.4 < reactionTime <= 1.5:
        print("40")
        bonusAchievement += 40
    elif 1.5 < reactionTime <= 1.7:
        print("30")
        bonusAchievement += 30
    else:
        print("0")
```

(12)

```python
    # Newline
    print("")

achievement = ((baseAchievement / TotalBasePointChart(noteSequence)) * 100) + (bonusAchievement / TotalBonusPointChart(noteSequence))

# Debug baseAchievement dan bonusAchievement
# print(f"Your base score is: {baseAchievement}")
# print(f"Your bonus score is: {bonusAchievement}")

# Print achievement
print("Congratulations on finishing the chart!")
print(f"Your score is: {achievement:.2f}%!\n")
```

(13)

The PlayChart function starts with initializing the achievements, both base and bonus. Then, looping is done for each note on the chart. Notes will be displayed (with a specific color according to the ColoredOutput function), and after a few seconds, the player will enter input. At the same time, there will be a timer that counts how long it takes the player to input. Please note that all time durations used in this function are examples only and do not correspond to the original maimai DX.

```python
# Dictionary notes, sebagai database notes
notes = {
    'T': {'good': 250, 'greatLow': 400, 'greatMid': 400, 'greatHigh': 400, 'perfectLow': 500, 'perfectHigh': 500, 'critical': 500},
    'HH': {'good': 500, 'greatLow': 800, 'greatMid': 800, 'greatHigh': 800, 'perfectLow': 1000, 'perfectHigh': 1000, 'critical': 1000},
    'SSS': {'good': 750, 'greatLow': 1200, 'greatMid': 1200, 'greatHigh': 1200, 'perfectLow': 1500, 'perfectHigh': 1500, 'critical': 1500},
    'BBBB' : {'good': 1000, 'greatLow': 1250, 'greatMid': 1500, 'greatHigh': 2000, 'perfectLow': 2500, 'perfectHigh': 2500, 'critical': 2500}
}
```

(14)

```python
# Hitung total score point
def TotalBasePointChart(chart) :
    basePoint = 0
    for i in range (len(chart)) :
        if (chart[i] == 'T') :
            basePoint += 500
        elif (chart[i] == 'HH') :
            basePoint += 1000
        elif (chart[i] == 'SSS') :
            basePoint += 1500
        elif (chart[i] == 'BBBB') :
            basePoint += 2500
    return basePoint
```

(15)

```python
def TotalBonusPointChart(chart) :
    bonusPoint = 0
    for i in range (len(chart)) :
        if (chart[i] == 'BBBB') :
            bonusPoint += 100
    return bonusPoint
```

(16)

Next, there is an if-else function which matches the duration of time the player enters input with a certain period of time, to produce score points corresponding to that duration. The same

thing is also done for break bonuses. Once again, the time duration used is just an assumption, what is in accordance with the original maimai DX is only the score points and calculations. The score points are then added to each achievement, and the input and calculation are repeated until all notes in the chart are processed.

Finally, total achievement is calculated using the formula stated in the previous chapter. The achievement is then displayed on the screen.

Below are some additional functions that do not have much effect on the main function or point calculation.

```
# Fungsi membuat chart random sementara sebagai pengganti chart lagu
def GenerateRandomList():
    noteSequence = ['T', 'HH', 'SSS', 'BBBB']
    randomChart = [random.choice(noteSequence) for _ in range(12)]
    print(randomChart)
```

(17)

(18)

```
# Fungsi pemberi warna supaya enak dipandang
def ColoredOutput(note):
    if note == 'T':
        return f"\033[38;2;200;0;150m{note}\033[0m"
    elif note == 'HH':
        return f"\033[38;2;255;182;193m{note}\033[0m"
    elif note == 'SSS':
        return f"\033[34m{note}\033[0m"
    elif note == 'BBBB':
        return f"\033[31m{note}\033[0m"
    else:
        return note
```

(19)

The PrintArt function is used to print Ascii Art (in Ascii Art it is written nainai DX and not maimai to avoid plagiarism). The GenerateRandomList function creates a random chart, namely an array of strings based on a certain number. This function is only used temporarily to create random charts. The ColoredOutput function is used to color notes to differentiate the notes on the chart.

C. Algorithm Complexity Calculations

For each function in the program, there will be time complexity analysis in T(n) and asymptotic time complexity analysis in Big-O notation (O(n)) will be analyzed where n is the number of input or data processed.

1. The GenerateRandomList() function executes the random.choice function 12 times (or whatever number is in range()). The random.choice function has complexity T(1) and O(1), so the total complexity is T(12) and O(12), which can be simplified to O(1). Interestingly, if the function is changed to accept n parameters which are the number of elements in the list, the complexity becomes T(n) and O(n).
2. The Coloredoutput(note) function changes notes to color based on the note type, with the time complexity Tmin = T(1) (note == 'T'), Tmax = T(5) (else option), and Tavg = T(2.5). Based on time complexity, this function has a complexity of O(1).
3. The LoadNotes(songChoice, diffChoice) function takes a chart in the form of a list from the global dictionary based on songChoice and diffChoice, so the time complexity is T(2), which means it has a complexity of O(1).
4. The TotalBasePointChart(chart) function calculates the maximum base point of a chart. Because there is repetition for each element in the chart array and comparisons occur between 1-4 times, for a chart of size n, the time complexity is Tmin = T(n), Tavg = T(2n), Tmax = T(4n), and the complexity is O(n). Note that there are only 4 possible notes so the last 'else if' part can be replaced with else, making Tmax = T(3n).
5. The TotalBonusPointChart(chart) function calculates the maximum bonus points from a chart. There is repetition for each element and a comparison occurs once, the time complexity is T(n) and the algorithm complexity is O(n).
6. The PlayChart(noteSequence) function iterates for each note in the chart/noteSequence and evaluates each player's input. For time complexity, Tmin is obtained from if all the notes are tapped (although it is possible that the program will not run because the number of breaks 0 divided by the total break score point 0 can fail the program) and all of them are critical, namely T(n), Tmax is obtained from if all the notes break and all fail (or all good) namely T(14n), and Tavg namely T(7n). Based on its time complexity, the complexity of this function is O(n).
7. The PrintArt() function only prints Ascii Art, so it has a time complexity of T(1) and a complexity of O(1).
8. Finally, the main function performs the maimai dx (PlayChart) game 4 times, so it has a time complexity of T(4) and algorithm complexity of O(1).

Based on the analysis above, we can take the algorithm complexity of the scoring system for a play on maimai DX from the greatest complexity, in this case it is O(n), or an algorithm whose implementation is linear, proportional to the number of inputs, in this case it is a lot of notes on the chart.

V. Conclusion

Maimai DX is one of the arcade rhythm games that is famous among young people. This game has various types of songs with their respective difficulty levels, where players can choose charts from Basic, Advanced, Expert, Master, and Re:Master difficulties. However, each difficulty has the same score point calculation. Base score points are calculated by adding up all the scores for each note based on the type of note (tap, hold, slide, break) and the accuracy of the player pressing the note (Critical, Perfect, etc.), divided by the maximum base score point from the chart, and multiplied by 100. Bonus score points are calculated by adding up all break bonuses (bonus points from note breaks) and dividing by the maximum break bonus from the chart. The player's score points on the chart, or achievement, are obtained by adding up the base score and bonus score.

Algorithmic complexity is a measure of how long an algorithm would take to complete given an input of size n, and is usually expressed with Big-O Notation. Based on the simple simulation created, the functions in a maimai DX game have algorithm complexity ranging from O(1) to O(n). This means that the algorithm complexity of maimai DX is O(n), where n is the number of notes on the chart being played.

V. Acknowledgement

I would like to thank God Almighty who has blessed me so that the work on this paper can run smoothly from start to finish. Then, I would like to thank my family, including my

parents, who have supported me in studying at ITB. I also thank all the lecturers and assistants, especially Mrs. Dr. Fariska Zakhralativa Ruskanda, S.T., M.T. for his guidance in studying the IF2120 Discrete Mathematics Subject. Lastly, I would like to thank my friends who have introduced me to the world of maimai, given me paper ideas and helped me work on this paper.

References

[1] Donmai. "Exploring the Algorithm Behind MaiMai DX's Scoring and DX Rating Computation". listed.io. June 26, 2003. [Online]. Available: https://listed.to/@donmai/45107/exploring-the-algorithm-behind-maimai-dx-s-scoring-and-dx-rating-computation. Accessed: Oct 6, 2023

[2] Munir, Rinaldi. (2023). Kompleksitas Algoritma (Bagian 1) [PowerPoint Slides]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/24-Kompleksitas-Algoritma-Bagian1-2023.pdf

[3] Munir, Rinaldi. (2023). Kompleksitas Algoritma (Bagian 2) [PowerPoint Slides]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/25-Kompleksitas-Algoritma-Bagian2-2023.pdf

[4] Sega. "maimai DX International Version Official Website". maimai.sega.com. [Online]. Available: https://maimai.sega.com/. Accessed: Oct 6, 2023.

[5] Upadhyay, Soni. "Time and Space complexity in Data Structure - Ultimate Guide". simplilearn.com. Oct 11, 2023. [Online]. Available: https://www.simplilearn.com/tutorials/data-structure-tutorial/time-and-space-complexity. Accessed: Dec 9, 2023.

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, 8 December 2023

Julian Caleb Simandjuntak (13522099)