Extended Essay

Computer Science

A Study on Software Design Approach

for Improving Trajectory Computation Time and Memory Usage
for Creating A Large Number of Shooting-Robots' Objects

Word Count : 3968

Pyoseon-High School
Senior, class 2, number 16

Chung, Hwee Joon (Chung, H. J.)

hweejoonchung@gmail.com
https://hweejoon-chung.github.io

Table of Contents

I. Introduction

1. Background

Not long ago, while enjoying a shooting game, a situation occurred in which the entire game operation was seriously slowed down or even some functions did not operate normally when the number of projectiles fired by robots in the game exceeded a certain number. Therefore, I want to recognize the performance degradation problem and try related analysis from the software point of view.

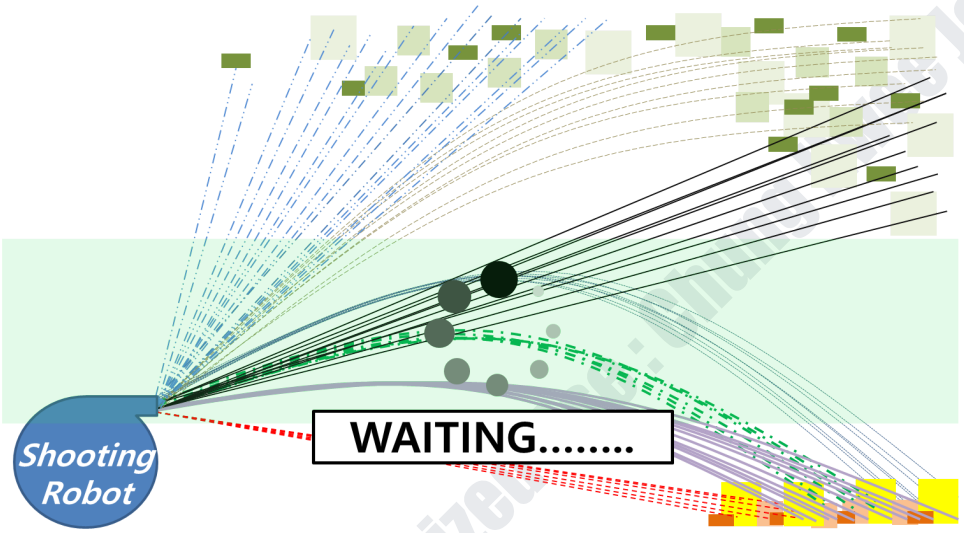


Figure 1. Speed/performance deterioration due to too many Projectiles

2. Objectives and Target Environment

Among several software quality characteristics defined by international standards (ISO/IEC, 2003) (ISO/IEC, 2016), the problem caused by performance degradation, which is the subject of this study, is related to software efficiency (Jung, 2007) (Yoon & Yu, 2021). In the field of software applications, there have been several studies on the factors affecting software efficiency, that is, software performance (Bahn, 2022) (EOM & MIN, 2013) (Kwon, 2018). As studies have shown, the main factors that greatly affect the efficiency quality of software are memory usage and CPU operation. In other words, an efficient memory management and calculation technique that can guarantee the quality of waiting time or response speed are required.

Setting a hypothesis on the problem in the shooting game earlier has the following considerations.

First, projectile objects occupy memory as they are created. That is to say, memory usage for generating multiple projectile objects may affect robot operation.

Second, each projectile object requires computation time for its trajectory. Namely the computation time for the trajectory of the projectile object may affect the robot motion.

As a result, it is judged that it is possible to analyze and verify the problem by checking the memory allocated to a large number of projectile objects and the computation time of each projectile trajectory. Therefore, in this study, I try to design/implement better software by checking the projectile trajectory computation time and projectile object generation memory usage from the software point of view independent of the game platform, and finding ways to improve it if possible.

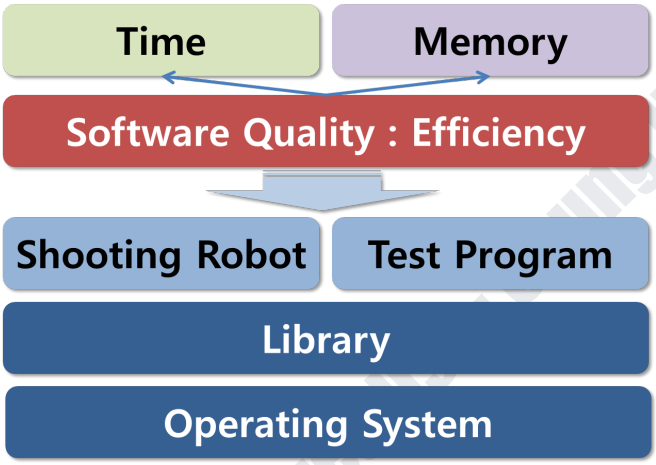


Figure 2. Software perspective Target environment considering Efficiency quality

Figure 2 shows the target environment of this study. For an approach from a software design point of view, a shooting robot can generate multiple projectiles, and the generated projectiles are launched through a projectile trajectory computation process according to various conditions. For the verification of this study, the GUI will not be implemented, I want to design and implement a shooting robot with a projectile and a launcher, a projectile that can have other properties, a launcher—for example, gun—to fire the projectile, calculating functions and testing functions for measuring time and memory.

3. Structure

The structure of this study is as follows. First, Chapter II describes related research. In Chapter III, I analyze the basic design of shooting robot and projectile trajectory computation techniques and derive multiple projectile object generation strategies from the object-oriented programming design perspective. In Chapter IV, I design/implement a test environment, and In Chapter V, I measure/compare changes in computation time and memory usage to verify improvement efficiency. Chapter VI presents conclusions and proposes future improvement tasks.

II. Related Research

This chapter first identifies the conditions and characteristics of software efficiency through research on software quality, builds background knowledge for establishing approaches or strategies available in this study based on projectile trajectory calculation and memory-related research, and analyzes object-oriented design and design patterns for software design/implementation.

1. Software Quality

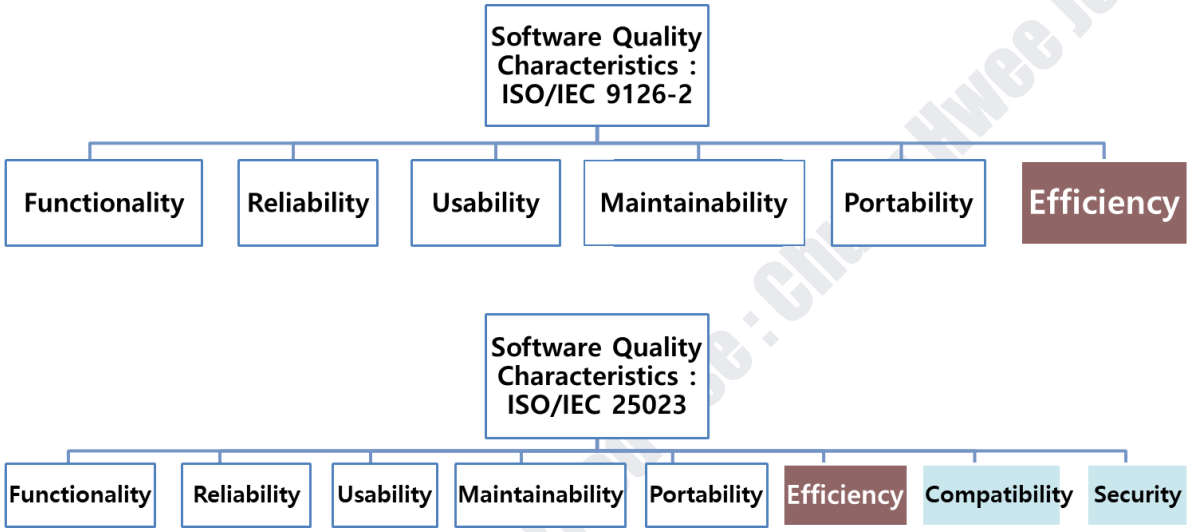


Figure 3. Quality characteristic model based on ISO/IEC 9126-2 and ISO/IEC 25023

Figure 3 (ISO/IEC, 2003) (ISO/IEC, 2016) is a software quality evaluation model defined by international standards. ISO/IEC 9126-2 and ISO/IEC 25023 define functionality, reliability, usability, maintainability, portability, and efficiency, including security and interoperability, as software quality characteristics. Among them, as shown in Table 1 (ISO/IEC, 2016), the use of time and resources—memory, CPU, I/O, etc.—are the main evaluation items for software efficiency quality related to this study.

Efficiency	Subcharacteristic	
	Time Behaviour	Response Time
		Throughput
		Turnaround Time
		…
	Resource Utilization	I/O Device Utilization
		Memory Utilization
		CPU Utilization
		…

	Capacity	No. of online requests
		…

Table 1. Efficiency testing based on ISO/IEC standards

In addition, among the quality of game software, that users demand the most is efficiency (Jung, 2007), which is the software's ability to provide the required performance according to the amount of resources used under prescribed conditions.

2. Projectile Trajectory Computation

The trajectory of a projectile can be calculated by calculus (Han, 2022), and the knowledge of calculus and simple ballistics applied in the world-famous game 'Angry Bird' (Song, 2020) can be used in my own existing studies, ‘ The most stable position of both hands using quadratic functions to improve juggling’(Chung, 2021) and ‘Derivating ballistic trajectory functions and implementing graph’(Chung, 2022).

In the study of (Chung, 2022), the parabolic—trajectory of the projectile—motion was divided into vertical motion and horizontal motion, and mathematical concepts such as differential, integral, and trigonometric functions were used for each motion to derive the positional functions of x-coordinate and y-coordinate for the trajectory of the projectile. In addition, in order to derive the position function for the expected arrival distance of a given projectile, the expected time of arrival, launch angle, initial velocity, etc. were calculated, and the trajectory graph of the launched projectile was expressed through entry programming in Figure 4.

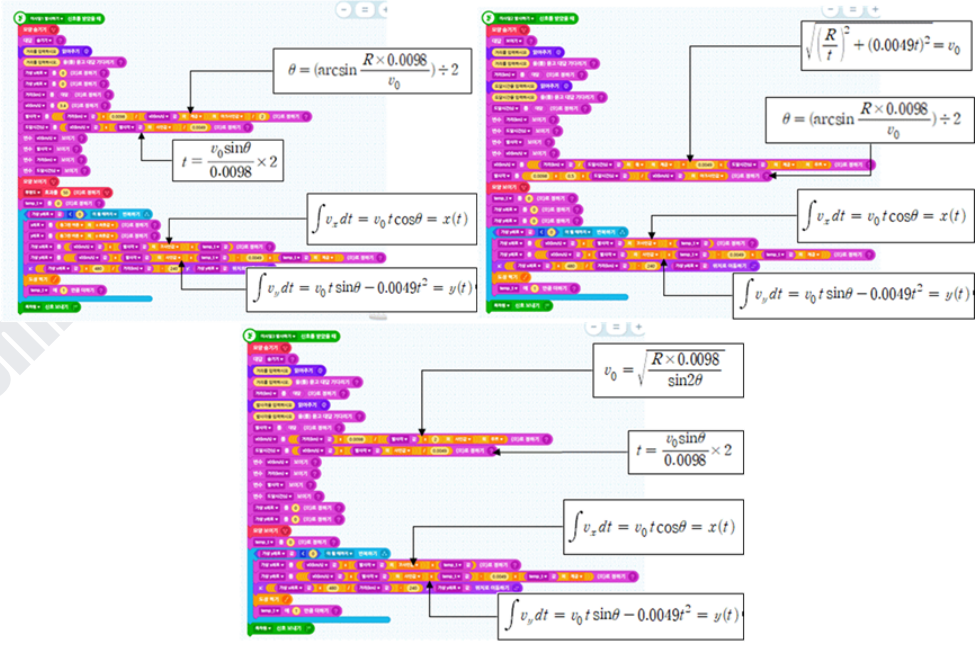


Figure 4. Entry Programming for Projectile trajectory computation

3. Memory Efficiency

Effective memory management and utilization techniques are needed to improve the quality of game software in an environment where resource use constraints exist. (Kim et al., 2007) Through this, game performance closely related to memory can be effectively improved.

In addition, there was also a study using a static memory management technique (Kim et al., 2009) to consider resources, that is, memory efficiency, when designing software.

4. Object-oriented Design Patterns

In order to increase productivity and quality of software, design techniques that can fully utilize object-oriented design concepts are useful when creating games. Figure 5(Kim, 2018) shows how to utilize design patterns of the software development process.

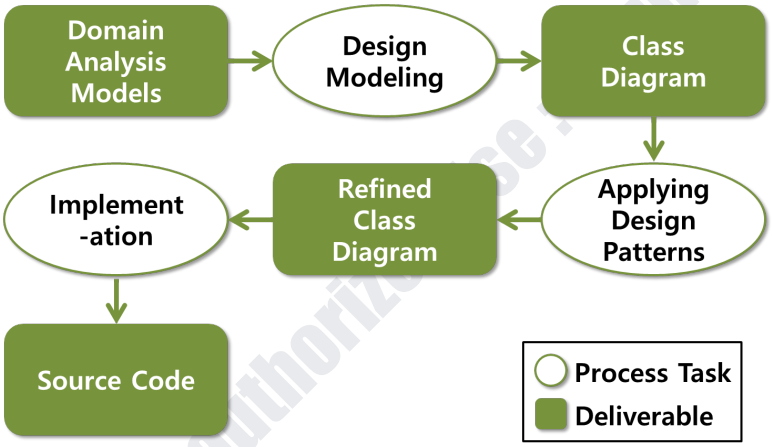


Figure 5. Utilizing Design patterns of software development process

In order to deal with the projectile entity in this study, I analyzed the creational pattern to abstract the creation procedure of the projectile entity among object-oriented design patterns and the structural pattern that expands to a larger structure. In general, there are five creational patterns and seven structural patterns: (Gamma et al., 1995)

Creational pattern:

(1) Abstract Factory: Groups other factories of the same class.
(2) Builders: create complex objects by separating creation and expression
(3) Factory method: You can create an object without specifying the class of the object to be created.
(4) Prototype: Creating an object by cloning an existing object.
(5) Singleton: There is only one instance of an object in a class.

Structural pattern:

(1) Adapter: Separates a class interface into another interface so other classes can use it
(2) Bridge: Separation of implementation and abstraction, each independently extensible
(3) Composite: It is possible to use both single objects and complex objects by configuring the relationship between objects in a tree form.
(4) Decorator: You can add other objects to one object to suit a specific situation or purpose.
(5) Facade: A wrapper interface can be provided by integrating sub-interfaces
(6) Flyweight: Possible to save memory by sharing as much as possible without randomly creating a large number of small-sized objects
(7) Proxy: Provide object surrogate or placeholder to control access to object

Object-oriented design is pursued by utilizing various design patterns (Gamma et al., 1995) in actual commercial games (Kim et al., 2007).

III. Analysis and Strategy Derivation

1. Basis

```
METHOD ShootingRobot::main()

BEGIN
      Ammunition : Object that are shot from Weapon
      AmmoFactory : Factory Object that generate Specific Ammunition
      Gun : One Object of Ranged Weapons that load with Ammunition
      numOfAmmo : Number of Ammunition

LOOP numOfAmmo
      Generate a Ammunition from AmmoFactory
      Set Specific Properties of the Ammunition
      Load Gun with the Ammunition
      Compute a ballistic trajectory of the Gun's Ammunition
Fire the Ammunition on the ballistic trajectory

END
```

Figure 6. Basic Algorithm

Figure 6 shows the basic algorithm of the shooting robot. After creating a projectile in a loop and setting its characteristics, it is loaded into the weapon and fired. At this time, the trajectory of the projectile is computed.

The definitions of the main terms used in this study are as follows.

Definition of term 1.
The two terms **Ammunition** and **Projectile** can be used interchangeably in this study.

Ammunition : objects that can be shot from a weapon, such as bullets or bombs
https://dictionary.cambridge.org/dictionary/english/ammunition
Projectile : an object that is thrown or shot forwards with force
https://dictionary.cambridge.org/dictionary/english/projectile

Definition of term 2.
The two terms **Entity** and **Object** can be used interchangeably in this study.
Entity : something that exists apart from other things, having its own independent existence
https://dictionary.cambridge.org/dictionary/english/entity
Object : In this study, Object is the meaning of Entity in the world of software.

2. Derivation of Formulas for computing Projectile Trajectories

In a shooting game, when a shooting robot fires a projectile, it is necessary to compute the path of the projectile. At this time, complex calculations are performed. Calculations using the concepts of differentiation and integration are required to calculate the angle, time of arrival, and trajectory of the projectile after receiving specifically the distance (Song, 2020), which affects the efficiency of the entire program in terms of time or speed.

Based on the previous research (Chung, 2022), this study assumes that the launch angle, initial velocity, arrival time, and reach distance are required to compute the projectile trajectory, and the shooting robot knows the reach distance among the four conditions. Therefore, if two or more of the three conditions except for the reach distance are given, computation formulas for obtaining three projectile trajectory coordinates can be derived—using trigonometric functions.

The motion of the projectile was divided into horizontal motion and vertical motion, and the horizontal motion was defined as constant velocity motion $v_x = v_0 \cos\theta$, and the vertical motion was defined as $v_y(t) = v_0 \sin\theta - 0.0098t$, considering that it was constant acceleration motion and was affected by gravity. The motion of the projectile is divided into horizontal and vertical motion, and the horizontal motion is expressed as $v_x = v_0 \cos\theta$ because it was assumed constant velocity motion, and the vertical is expressed as $v_y(t) = v_0 \sin\theta - 0.0098t$ because it was a constant acceleration motion under the influence of gravity. In general, the gravitational acceleration is $g = 9.8 m/s^2$, but the unit of the projectile speed is usually $Mach$, which is $1\ Mach \approx 0.34 km/s$, so the unit of the

gravitational acceleration is also changed to km/s^2, and the gravitational acceleration is $g = 0.0098km/s^2$.

In this study, I try to select the most optimal formula by comparing the amount of calculation—computation time—of the formulas to obtain three types of projectile trajectory coordinates.

In order to obtain a ballistic trajectory, a function of x and y coordinates for t —hereinafter, projectile trajectory coordinate function—can be obtained as in

$$x(t) = \int v_x dt = \int v_0 cos\theta dt = v_0 cos\theta t + C \approx v_0 cos\theta t$$

$$y(t) = \int v_y(t) dt = \int (v_0 sin\theta - 0.0098t) dt = v_0 sin\theta t - 0.0049t^2 + C \approx v_0 sin\theta t - 0.0049t^2$$

At this time, assuming that the reach distance R is fixed, the arrival time T, the launch angle θ, and the initial velocity v_0 become variables in the computation formula, and the following three computation formulas are derived.

The first formula receives the reach distance R, calculates the expected arrival time T and the launch angle θ, and returns the x, y coordinates. Here, the initial speed v_0 was set to $10Mach—3.4km/s^2$. First of all, it is necessary to obtain the maximum altitude H and t_H. The former is the maximum value of y, the latter makes the differential coefficient of y for time t equal to zero.

t that satisfies

$$\frac{dy}{dt} = v_0 sin\theta - 0.0098t = 0$$

is a variable that has the highest altitude H, namely the maximum value of y, so let's call it t_H .

$$t_H = \frac{v_0 sin\theta}{0.0098}, \quad H = \frac{v_0^2 sin^2\theta}{0.0049}$$

In general, the time to reach the highest altitude in parabolic motion is half of the total motion time of the parabolic motion body, so the time T to reach the motion body is twice the time t_H to reach the highest altitude H.

$$\therefore T = 2t_H = \frac{v_0 sin\theta}{0.0049}$$

In addition, the reach distance R is expressed as follows using the double angle formula — $sin2\theta = 2sin\theta cos\theta$ — of the trigonometric function. $R = x(T) = \frac{v_0^2 sin\theta cos\theta}{0.0049} = \frac{v_0^2 sin2\theta}{0.0098}$

Also, the launch angle θ was rearranged using the inverse trigonometric function in the above equation to obtain the reach distance *R*.

$$\therefore \theta = \left(arcsin\frac{R \times 0.0098}{v_0^2}\right) \div 2$$

By substituting the calculated *T* & θ, into the projectile trajectory coordinate function, the function of x, y coordinates with respect to time t can be obtained as follows.

$$x(t) = v_0 cos\theta t = v_0 cos\left\{\left(arcsin\frac{R \times 0.0098}{v_0^2}\right) \div 2\right\} \times t$$

$$y(t) = v_0 sin\theta t - 0.0049t^2 = v_0 sin\left\{\left(arcsin\frac{R \times 0.0098}{v_0^2}\right) \div 2\right\} \times t \ - 0.0049 \times t^2$$

The second formula receives the arrival distance *R* and the arrival time *T*, calculates the initial velocity v_0 and the launch angle θ, and returns the x, y coordinates. The launch angle θ uses the result derived from the first formula. $\therefore \theta = \left(arcsin\frac{R \times 0.0098}{v_0^2}\right) \div 2$

Since the arrival time t is twice the time to reach the highest altitude *H*,

$$T = \frac{v_0 sin\theta}{0.0098} \times 2 = \frac{v_0 sin\theta}{0.0049} \qquad \therefore v_0 sin\theta = 0.0049 \times T$$

Using this expression, substituting for $R = \frac{v_0^2 sin\theta cos\theta}{0.0049}$ yields $v_0 cos\theta = \frac{R}{T}$.

$$\left(\frac{R}{T}\right)^2 + (0.0049T)^2 = v_0^2 \qquad \left(\because v_0^2 \, sin^2\theta + v_0^2 cos^2\theta = \ v_0^2 \right)$$

$$\therefore v_0 = \sqrt{\left(\frac{R}{T}\right)^2 + (0.0049T)^2}$$

By substituting the above-obtained θ and v_0 into the projectile trajectory coordinate functions, the function of x, y coordinates with respect to time t can be obtained as follows.

$$x(t) = v_0 cos\theta t = \sqrt{\left(\frac{R}{T}\right)^2 + (0.0049T)^2} \times cos\left\{\left(arcsin\frac{R \times 0.0098}{v_0^2}\right) \div 2\right\} \times t$$

$$y(t) = v_0 sin\theta t - 0.0049t^2 = \sqrt{\left(\frac{R}{T}\right)^2 + (0.0049T)^2} \times sin\left\{\left(arcsin\frac{R \times 0.0098}{v_0^2}\right) \div 2\right\} \times t - 0.0049t^2$$

The third formula receives the reach distance R and the launch angle θ, calculates the initial velocity v_0 and the expected arrival time T, and returns the x, y coordinates.

Since $R = \frac{v_0^2 sin\theta cos\theta}{0.0049} = \frac{v_0^2 sin2\theta}{0.0098}$ in the first formula,

$$\therefore v_0 = \sqrt{\frac{R \times 0.0098}{sin2\theta}}$$

The arrival time T uses the result derived from the first formula. $T = \frac{v_0 sin\theta}{0.0098} \times 2 = \frac{v_0 sin\theta}{0.0049}$

By substituting the obtained v_0 and T into the projectile trajectory coordinate functions, the function of x, y coordinates with respect to time t can be obtained as follows.

$$x(t) = v_0 cos\theta t = \sqrt{\frac{R \times 0.0098}{sin2\theta}} \times cos\theta t$$

$$y(t) = v_0 sin\theta t - 0.0049t^2 = \sqrt{\frac{R \times 0.0098}{sin2\theta}} \times sin\theta t - 0.0049t^2$$

In this study, the three computation formulas derived above are used to calculate the projectile trajectory.

3. Analysis and Selection of Object Creation Method

In software development, the most basic object creation method is to receive the number of projectile objects as input and create the corresponding number of projectile objects. If you create multiple projectiles, the memory usage increases proportionally.

In order to prepare a strategy for improving the problem of the basic object creation method, the characteristics of projectile object creation from the object-oriented design point of view are as follows.

(1) Many identical projectile objects spawn independently
(2) Projectiles may have different properties depending on the launcher.
(3) Creation constraints of projectile objects depend on the system on which the software is running.

The strategy derived from this study to improve the above characteristics among object-oriented design methods is as follows.

(1) Use design patterns to create them independently.
(2) A projectile object needs a factory class to enable multiple projectile types when created.
(3) The Flyweight pattern (Gamma et al., 1995) is applied to create a projectile object for simple projectile trajectory expression without specific state values—Figure 7.

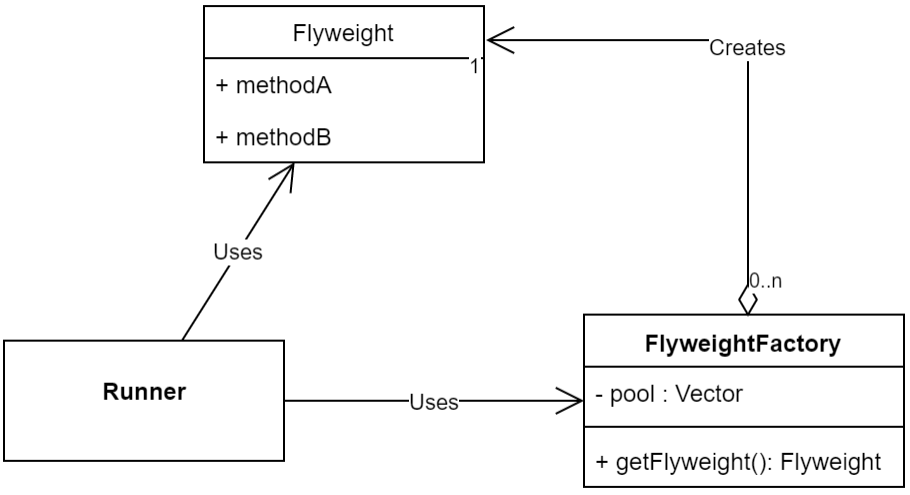


Figure 7. Flyweight Design pattern

4. Design with Improvement Strategy

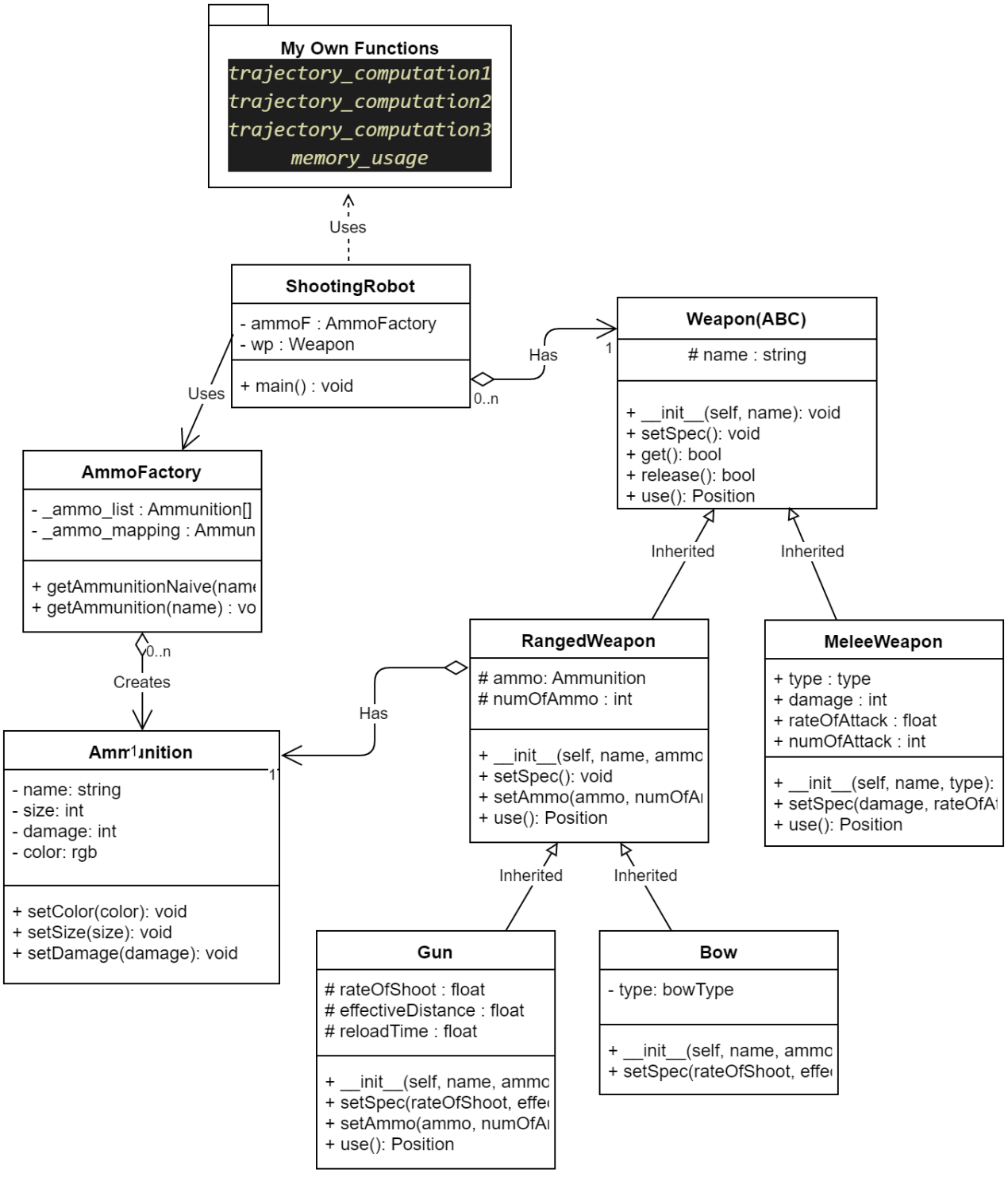


Figure 8. Designed Class Diagram for Shooting robot

Figure 8 is a class diagram of the shooting robot designed in this study.

The top-level abstract class of the weapon used by the shooting robot is Weapon, and has the RangedWeapon class that handles long-range weapons and the MeleeWeapon class that handles short-range weapons as children classes. The RangedWeapon class has the Gun class that handles guns used in this study and the Bow class that handles bows as children classes.

In this study, the Ammunition class that handles projectiles mounted on the gun used by the shooting robot and the AmmoFactory class that creates the desired ammunition were designed. When creating Ammunition, the Flyweight pattern was applied among the design patterns.

In addition, functions for trajectory computation and a function for measuring memory usage are designed and used for verification of this study.

IV. Implementation

1. Overall System Structure

The figure shows the implementation environment and program structure of this study. The implementation environment utilized Visual Studio Code IDE in the Windows 10 environment and selected the most popular python as the programming language. The actual Python version used was v.3.11.2, and as libraries, numpy for trigonometric functions, math for root operations, psutil for memory-related functions, and timeit for time functions were used.

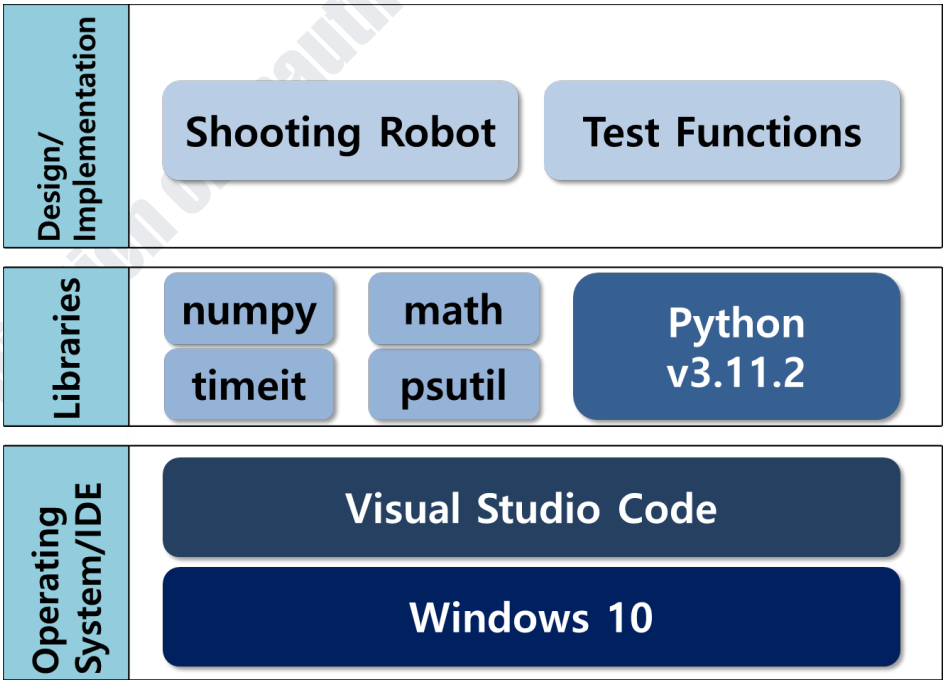


Figure 9. target environment

2. Implementation of Formulas for Projectile Trajectory Computation

I compare the actual computation amount by implementing three formulas for calculating the projectile trajectory derived above. Since the GUI is not implemented, the computation is close to the actual amount of calculation, but when the GUI is implemented, the overhead for graphics becomes larger.

A brief summary of the three computation formulas is shown in Table 2 below.

[Trajectory Computation#1]	A function that calculates the x,y coordinates of a projectile with respect to time by receiving the expected arrival distance and initial velocity of the projectile
[Trajectory Computation#2]	A function that calculates the x,y coordinates of a projectile with respect to time by receiving the expected arrival distance and expected arrival time of the projectile
[Trajectory Computation#3]	A function that calculates the x,y coordinates of a projectile with respect to time by receiving the expected arrival distance and launch angle of the projectile

Table 2. Formulas for performing projectile trajectory computation

```
def trajectory_computation1(R, v0) :  #[Trajectory Computation#1]
    theta = np.arcsin(R*0.0098/v0**2)/2
    T = v0*np.sin(theta)*2/0.0098
    for t in range(T):
        x = v0*np.cos(theta)*t
        y = v0*np.sin(theta)*t - 0.0049*t**2
    return x, y, T
```

```
def trajectory_computation2(R, T) :   #[Trajectory Computation#2]
    v0 = math.sqrt((R/T)**2 + (0.0049*T)**2)
    theta = np.arcsin(R*0.0098/v0**2)/2
    for t in range(T):
        x = v0*np.cos(theta)*t
        y = v0*np.sin(theta)*t - 0.0049*t**2
    return x, y, T
```

```
def trajectory_computation3(R, theta) :#[Trajectory Computation#3]
    v0 = math.sqrt(R*0.0098/np.sin(2*theta))
    T = v0*np.sin(theta)*2/0.0098
    for t in range(T):
        x = v0*np.cos(theta)*t
        y = v0*np.sin(theta)*t - 0.0049*t**2
    return x, y, T
```

The above codes are functions actually implemented in Python language by using various libraries.

3. Implementation of Pattern-based Projectile Generation

In general, objects for projectiles are created and used as many times as necessary. I compare the usage of each memory for the implementation of the object creation technique designed in this study and the implementation of the general object creation method. A smaller actual memory usage can be seen as a method with higher efficiency among software quality characteristics.

METHOD getAmmunitionNaively ***BEGIN*** ***Ammunition : Object that are shot from Weapon*** ***AmmoFactory : Factory that generate Specific Ammunition*** ***Ammunition List: List Data Structure for managing Ammunitions*** ***numOfAmmo : Number of Ammunition*** ***LOOP numOfAmmo*** ***Generate a Ammunition from AmmoFactory*** ***Add the Ammunition to Ammunition List*** ***END***	***METHOD getAmmunition*** ***BEGIN*** ***Ammunition : Object that are shot from Weapon*** ***AmmoFactory : Factory that generate Specific Ammunition*** ***Ammunition Map : Map Data Structure for managing Ammunitions*** ***numOfAmmo : Number of Ammunition*** ***LOOP numOfAmmo*** ***Get a Ammunition from Ammunition Map*** ***if ammunition is none*** ***Generate a Ammunition from AmmoFactory*** ***Add the Ammunition to Ammunition List*** ***END***

Figure 10. General object creation method and creation technique algorithm designed in this study

Figure 10 shows the basic algorithm for the general entity generation method —hereinafter entity generation method #1—and the generation technique designed in this study—hereinafter entity generation method #2. The difference between the two methods is

whether memory is dynamically allocated or statically allocated memory is used when needed.

4. Experimental Test

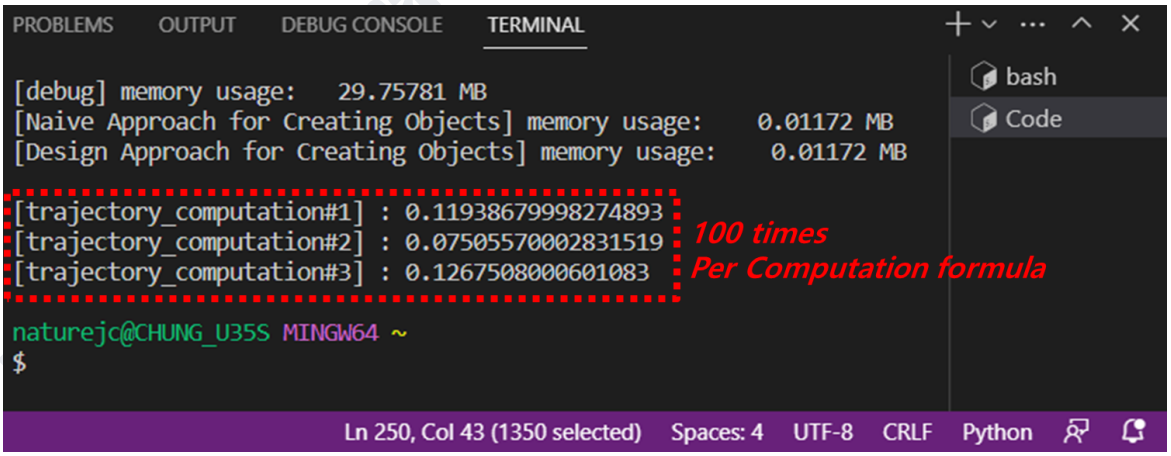
For the implementation experiment, 100, 1000, and 10000 test projectile objects were created and tested. The shooting robot implemented for this study operates based on text, and the settings for projectiles are stored and used in static form within the program code, such as name, color, size, damage, and number. In addition, in order to create and use various projectiles, it is assumed that the launching device of the shooting robot is ComplexGun.—see code below

```
names = ["Pellet", "Bullet", "Projectile", "Arrow"]
colors = ["Red", "Yellow", "Green", "Blue"]
sizes = [10, 30, 60, 40]
damages = [3, 5, 8, 1]
nums = [9, 5, 3, 10]

gun = Gun("ComplexGun")
numOfShoot = NUM_OF_SHOOT    # 1000 | 10000 | 100000
```

V. Verification

1. Comparison of Formulas for Projectile Trajectory Computation



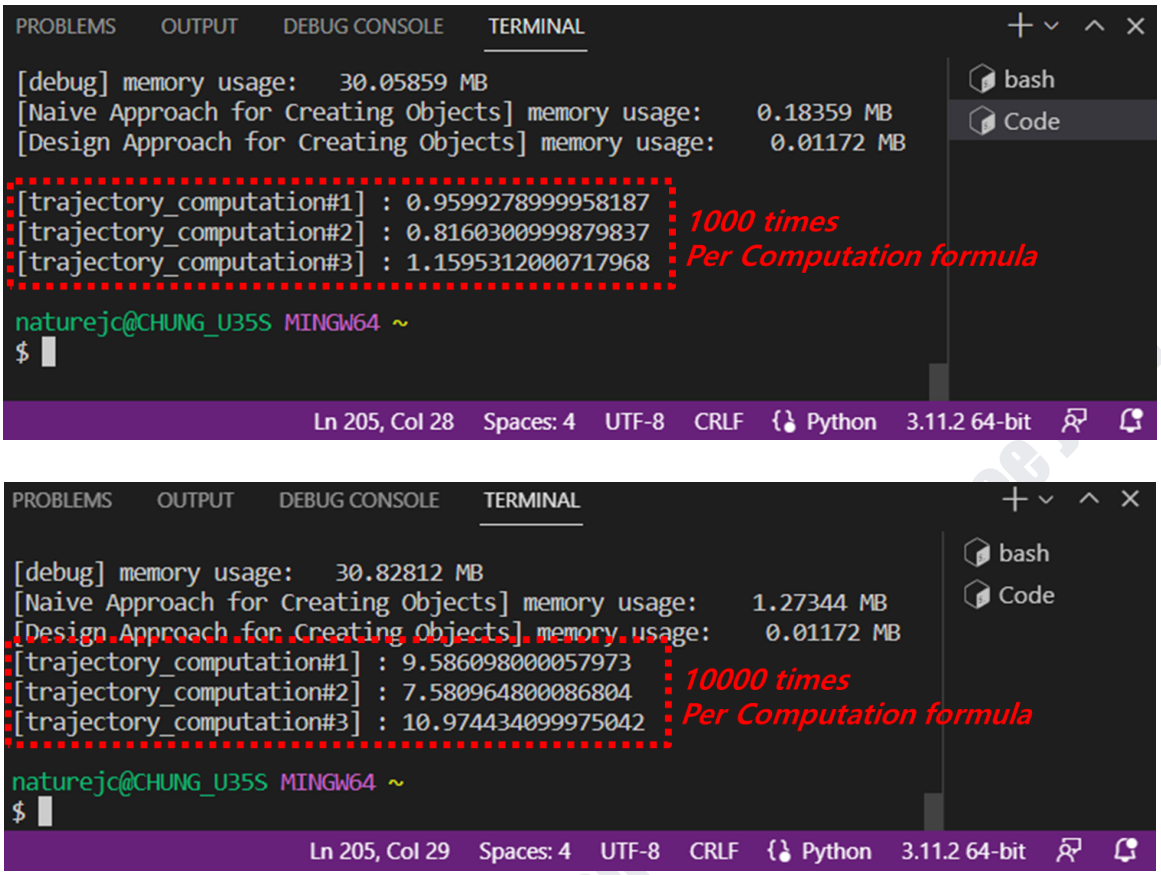


Figure 11. Computation time of 3 formulas for projectile trajectory

Figure 11 shows the result of programming the cumulative computational time when three formulas for the projectile orbit are executed 10, 100, and 1000 times.

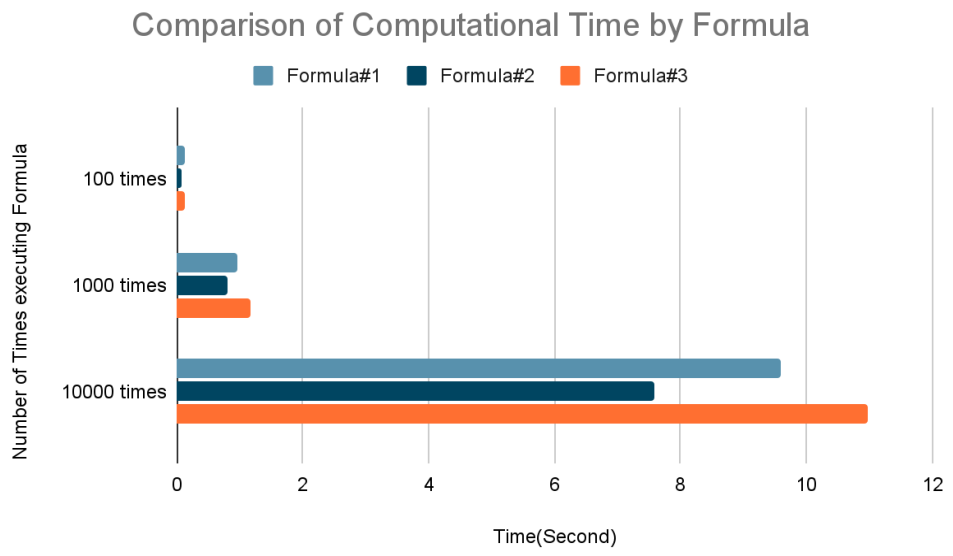


Figure 12. Comparison of Computational Time by Formula for projectile trajectory

The degree of efficiency among the software quality characteristics was determined by measuring the time for the three computation formulas for the projectile trajectory derived above. Although the difference between the three formulas is not very large, as the result—Figure 12—, the most efficient formula in the implementation test was the second that calculates the x,y coordinates of the projectile with respect to time by receiving the expected arrival distance and expected arrival time of the projectile, that is tragectory_computation#2.

2. Comparison of Memory Usage for Object Creation

In terms of implementation, test operations that create and set objects are shown in Figure 13 below. The actual memory usage is calculated by analyzing the difference in memory usage before and after creating objects.

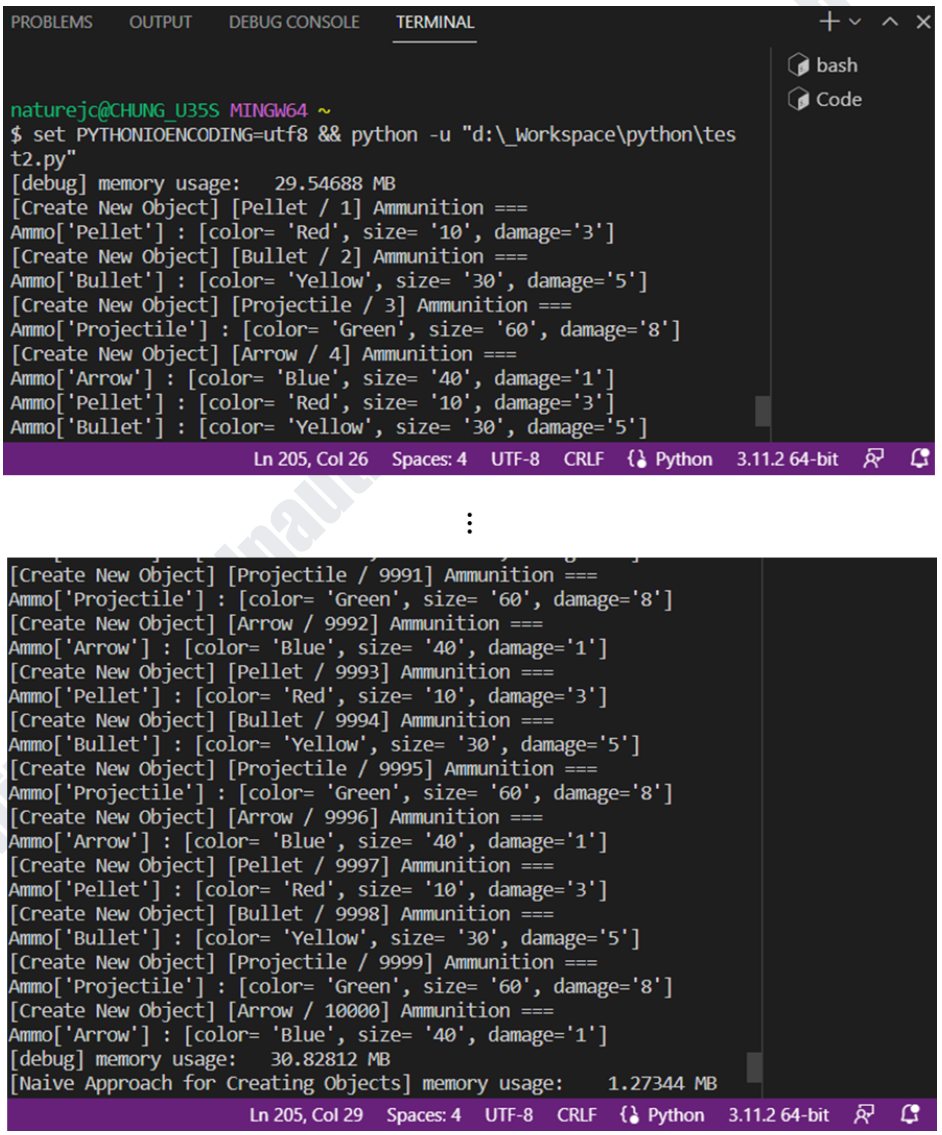


Figure 13. implementation test behavior

[Naive Approach for Creating Object] memory usage on the test result screen is the memory usage when memory needs to be allocated whenever a general object is created, and **[Design Approach for Creating Object]** memory usage shows the pattern-based creation technique designed above. This is the memory usage when creating multiple objects using

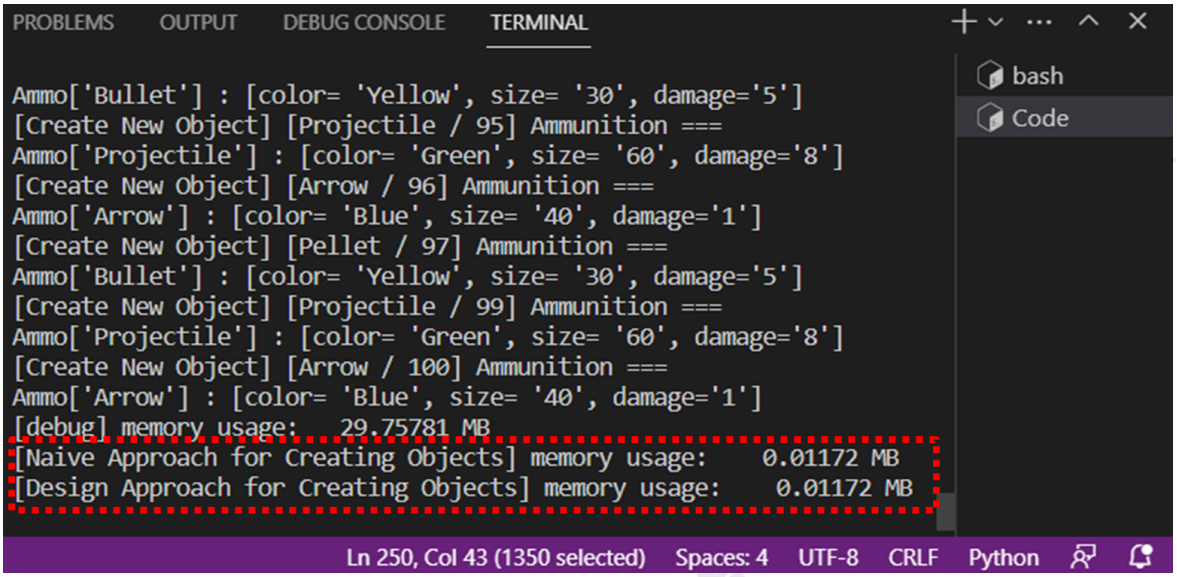


Figure 14. Memory usage when creating 100 projectile objects

Figure 14 shows the experimental results for 100 projectile objects. The memory usage of the general creating method and the creating method designed in this study is almost the same.

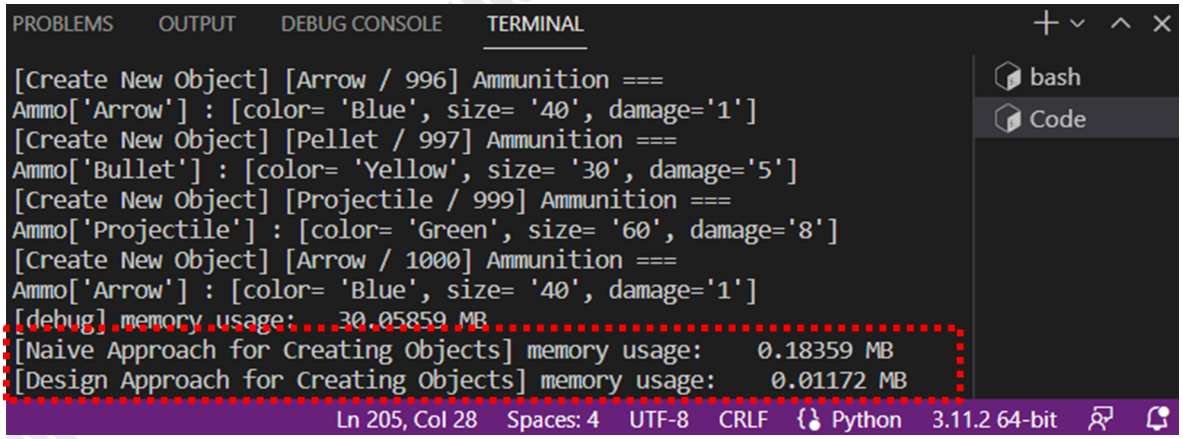


Figure 15. Memory usage when creating 1000 projectile objects

Figure 15 shows the experimental results for 1000 projectile objects. It can be seen that the memory usage of the creating method designed in this study is only 6% of that of the general creating method.

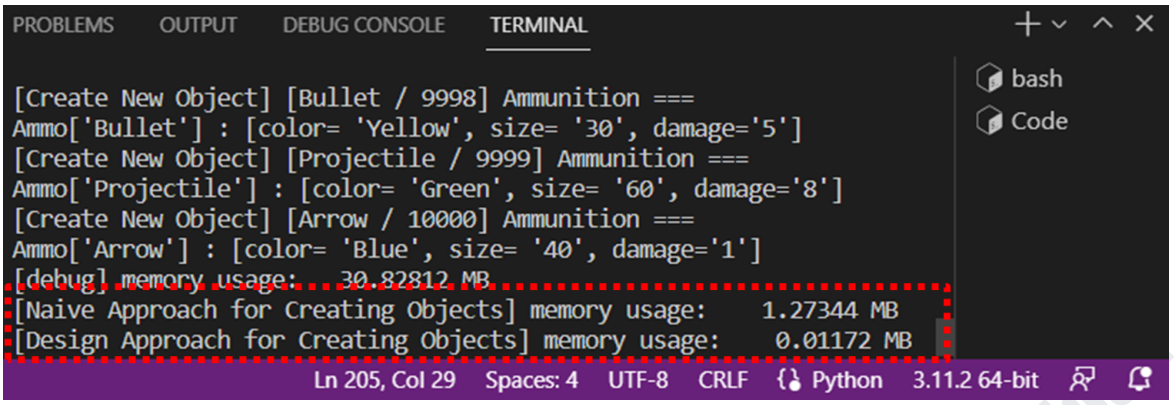


Figure 16. Memory usage when creating 10000 projectile objects

Figure 16 shows the experimental results for 10000 projectile objects. It was found that the creating method designed in this study uses only 0.9% of the memory compared to the general creating method.

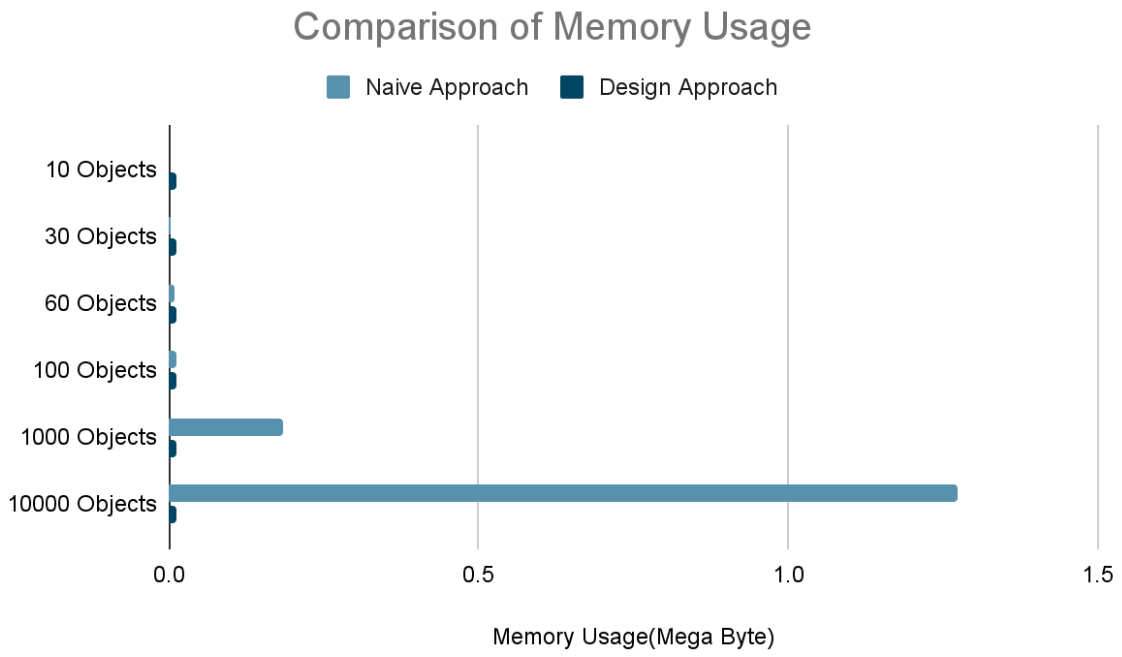


Figure 17. Comparison of Memory Usage

Looking at the graph in Figure 17, there is not much difference in memory usage under 60 projectile objects, but if the number of projectile objects exceeds 100 units, the memory usage of the designed creation technique is much more efficient.

Although it cannot be said to be perfect because it is data on a simple test program to which GUI, other functions and processes are not applied, it was verified that more efficient memory can be used when creating multiple projectile objects using the generation technique designed in this study.

VI. Conclusion and Future Tasks

This study deals with the overhead problem that occurs when shooting multiple projectiles in a shooting robot. From the point of view of platform-independent software, the presented problem was improved through the following process.

(1) The problem of generating multiple projectile overhead was designed in an object-oriented way and implemented in the python programming language.
(2) The projectile generation and projectile trajectory computations were analyzed to implement the memory amount and time measurement function.
(3) In order to improve the overhead problem, a creation method of flyweight pattern-based projectile entity and a comparison technique of projectile trajectory computation formulas were designed.
(4) Memory usage and computation time for each design were measured and compared.

In addition, in a related study conducted by this researcher (Chung, 2022), errors in mathematical calculation equations were found and corrected during the course of this study.

Since the limitations of this study were tested from the perspective of software design, it was verified in a test environment, not in a situation where the actual shooting robots were working. And only the shooting robots were designed and verified, so design and implementation of the entire software can be carried out later.

Works Cited

Bahn, H. (2022). End-to-End Resource Management Techniques for Supporting Real-time Tasks in Mobile Devices. *The Journal of the Institute of Internet, Broadcasting and Communication*, *22*(5), 43-48. https://doi.org/10.7236/JIIBC.2022.22.5.43

Chung, H.J. (2021). 저글링을 할 때 이차함수를 이용하여 가장 안정적인 양손의 위치. HWEE JOON's Persona. https://hweejoon-chung.github.io/mathreport202106-assessment

Chung, H.J. (2022). 탄도(탄 궤적) 함수 유도 및 그래프 구현. HWEE JOON's Persona. https://hweejoon-chung.github.io/calculusTrajectory-assessmentMATH

EOM, Y. I., & MIN, C. (2013). Resource Management Framework for Improving Thread Performance on Mobile Platform. *Journal of KIISE:Computer Systems and Theory*, *40*(5), 207-216.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns*. Addison-Wesley.

Han, H. (2022). 미적분의 쓸모. 더퀘스트.

ISO/IEC. (2003). *ISO/IEC TR 9126-2 Software engineering - Product quality*.

ISO/IEC. (2016). *ISO/IEC 25023 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Measurement of system and software product quality*.

Jung, H.-J. (2007, 12). An Evaluation Model of Game Software. *Journal of Internet Computing and Services*, *8*(6), 115-125.

Kim, D. K. (2018). Design Patterns for Android Game Programming. *Journal of the Korea Convergence Society*, *9*(8), 17-24. https://doi.org/10.15207/JKCS.2018.9.8.017

Kim, H.-Y., Ham, D.-H., & Kim, M.-S. (2009). A Study of Object Pooling Scheme for Efficient Online Gaming Server. *Journal of Korea Game Society*, *9*(6), 163-170.

Kim, T.-I., Kim, S.-G., & Han, H.-S. (2007). Reusing Local Regions in Memory-limited Java Virtual Machines. *Journal of KIISE:Software and Applications*, *34*(6), 562-571.

Kim, T.-S., Kim, S.-H., & Kim, J.-S. (2007). The Item Distribution Method for the Party Systemin the MMORPG Using the Observer Pattern. *JOURNAL OF KOREA MULTIMEDIA SOCIETY*, *10*(8), 1060-1067.

Kwon, O.-S. (2018). Comparison of System Resource Usage for Screen Image Transmission. *Journal of Knowledge Information Technology and Systems*, *13*(1), 129-137. http://doi.org/10.34163/jkits.2018.13.1.013

Song, H. (2020). 이렇게 흘러가는 세상. MID.

Yoon, G., & Yu, J. (2021). Applying ISO/IEC 25023 to Software Engineering Process in Weapon System for Quality Improvement. *Journal of the Korea Academia-Industrial cooperation Society*, *22*(5), 387-393. https://doi.org/10.5762/KAIS.2021.22.5.387