Gacha Game Simulator

Object Based Implementation

Deadlines: MCO1A Initial Submission: April 6, 2021 (7:59 PM)
MCO1A Final Submission, MCO1B: April 19, 2021 (7:59 PM)

I. Project Overview

Gacha games have risen in popularity in recent years, with gacha (and to some extent loot boxes) becoming the de facto monetization strategy used by freemium games or games as services. The gacha system involves players spending an amount of in-game currency (usually free) to initiate a random draw (colloquially called a **pull**) from a pool of useful game entities (often characters or weapons, but other entities have been obtainable using gacha as well) from the **gacha machine** (the term used to call the part of the game that generates the random draws). Players, however, are incentivized to do a **10-pull**, basically instead of pulling individually, pulling only when the player has enough in-game currency to pull 10 times. This results in either a guaranteed rare (and often highly desirable) game entity, or a discounted pull price.

The game entities are often used in other aspects of the game, and powerful entities are often rarer and harder to get from the gacha machine. Players hoping to pull a specific game entity may end up having a surplus of unwanted or otherwise more common entities that can be obtained from the gacha machine. Games that employ a gacha system would often have a way to recycle these unwanted or redundant game entities, by **merging** them together (sometimes the process is called **refining** or **upgrading**, which is a different process from **levelling**). Often, three of the same kind of a game entity may be merged to get a more powerful or more desirable version of the entity, often with stronger overall properties. Game entities may also be **levelled** using a different levelling resource, which results in a smaller but steadier increase in power and desirability for the game entity.

For this project, you will be making a gacha simulator. You are to make several entities that model common tropes found in gacha games, namely:

1. A **Gacha machine** that facilitates pulling of characters and weapons
2. **Characters** that are used to combat enemies; duplicate characters may be merged together so they may be refined
3. **Weapons** that give characters additional power; duplicate weapons may be merged together so they may be refined
4. **Resources** which are used to level both characters and weapons
5. **Enemies** that characters can defeat, which yields resources
6. **Maps** where different kinds of enemies are found
7. Other **managers** that facilitate merging, levelling, combat and exploration.

Each of these entities would be discussed in detail in their own section. Do note that while this project is meant to simulate a gacha game, its systems do not follow any particular gacha game. Please make sure to read through the specifications to know exactly what is expected of you. Also note that this first phase of the machine project yields an **object-based** implementation; after the midterms when you learn inheritance, superclasses and subclasses, you will further refine this implementation and make it truly **object-oriented**. Finally, a GUI is not required for the first phase of the project.

II. Characters

Characters in this project are obtained from the Gacha Machine, and has (but is not limited to) the following properties:

1. Name - A string that serves as the label for a character. As mentioned in the project overview, since players are expected to pull duplicates of the same character, characters may have the same name.
2. Rarity - An integer value from 1 to 5 which determines how common the character is to get from the Gacha Machine. Each character pull has a 50% chance of being a 1-rarity character, a 35% chance of being a 2-rarity character and a 15% chance of being a 3-rarity character. This value is used in calculating adventure success and the amount of rewards obtained from the adventure.
3. Element - defines the type of the character. As would be mentioned later in the System Managers section, two characters can be sent to explore any given region, and depending on the element pairing, certain effects will be triggered. See the table of Type Combinations in the Appendix for more information, but the elements are:
    a) Joker - a well-rounded type that pairs well with Cyclone, but can also pair well with Heat and Luna. Pairing with Trigger or Metal is possible but has a generally weak effect.
    b) Trigger - a primarily ranged type, pairs best with Luna but has no effect when paired with Heat, Cyclone, Joker or Metal. Pairing with the latter four elements is still possible, however.
    c) Metal - a primarily melee type, pairs great with Heat, but can also pair well with Cyclone and Luna.
    d) Cyclone - a wind-based type. Pairs best with Joker, but can pair with Trigger and Metal as well, though pairing with Trigger has no effect. Causes negative effects when paired with Heat or Luna.
    e) Luna - a light-based type. Pairs best with Trigger, but can pair with Joker and Metal as well. Causes negative effects when paired with Heat or Cyclone.
    f) Heat - a fire-based type. Pairs best with Metal, but can also be paired with Joker and Trigger. Causes negative effects when paired with Luna or Cyclone.
4. Level - An integer value that always starts at 20, with a maximum value of 100. This value is used in calculating adventure success and the amount of rewards obtained from the adventure.
5. Weapon - A Weapon object, a character can only have one weapon equipped. Also, a weapon can only be equipped to just one character.

See **Appendix B: Character List** for a full list of characters and their attributes.

Characters can:

1. Be merged with two other characters that bear the same name and rarity. Once merged, the other two characters used for merging are consumed, and the character that was merged into increases its rarity value by 1 (remember that the limit is 5, merging should no longer be possible after reaching rarity 5). A Refining Manager may be necessary to facilitate this merging process.
2. Be levelled up using a resource instance. For the first phase of the machine project, a single resource (simply called resource) is used to level up characters. A character's level may be increased by one by consuming one resource instance. It should be possible for players to use multiple resource instances in one levelling action, thus increasing the level of the character by one an equal number of times. A Levelling Manager may be necessary to facilitate the levelling process.
3. Equip a weapon. As mentioned, once a weapon is equipped, it cannot be equipped to any other character, or more accurately, if the weapon is already equipped, it is removed from the character currently using the weapon before being transferred to the character it is being equipped to.
4. Go on an adventure. Along with another character, a pair of characters may be deployed into a map to explore and fight against enemies. Fighting enemies should be automatically done by the program; **there is no need for any combat system for this project**. Depending on the element combination, rarities, levels and weapons equipped by each character, adventures would yield certain amounts of resources. An Adventure Manager may be necessary to facilitate this adventure process.

III. Weapons

Weapons in this project gives characters power, and determines the success of adventures characters are deployed under. A successfully completed adventure levels the characters involved in the adventure by 1, while an excellently completed adventure levels the characters involved in the adventure by 2. Weapons can also be obtained from the Gacha Machine, and has (but is not limited to) the following properties.

1. Name - A string that serves as the label for a weapon. As mentioned in the project overview, since players are expected to pull duplicates of the same weapon, weapons may have the same name.
2. Power - An integer that determines the amount of power the weapon confers to the character equipping the weapon. This power is scaled up or down by some of the other attributes of the weapon in computations involving character power. The lowest possible power is 130 while the highest possible power is 230.
3. Rarity - An integer value from 1 to 5 which determines how common the weapon is to get from the Gacha Machine. Each weapon pull has a 50% chance of being a 1-rarity weapon, a 35% chance of being a 2-rarity weapon and a 15% chance of

being a 3-rarity weapon. Depending on the rarity, the weapon's power is scaled via a multiplier:
    a) 1-rarity weapons get a multiplier of 0.7 (70%) as their rarity multiplier
    b) 2-rarity weapons get a multiplier of 0.8 (80%) as their rarity multiplier
    c) 3-rarity weapons get a multiplier of 0.9 (90%) as their rarity multiplier
    d) 4-rarity weapons get a multiplier of 1.0 (100%) as their rarity multiplier
    e) 5-rarity weapons get a multiplier of 1.2 (120%) as their rarity multiplier

4. Level - An integer value that always starts at 1 and ends at 50. This is an additive bonus to the power of the weapon, and is used in calculating adventure success and the amount of rewards obtained from the adventure.

See **Appendix C: Weapon List** for a full list of weapons and their attributes.

Weapons can:

1. Be merged with two other weapons that bear the same name and rarity. Once merged, the other two weapons used for merging are consumed, and the weapon that was merged into increases its rarity value by 1 (remember that the limit is 5, merging should no longer be possible after reaching rarity 5). A Refining Manager may be necessary to facilitate this merging process.
2. Be levelled up using a resource instance. For the first phase of the machine project, a single resource (simply called resource) is used to level up a weapon. A weapon's level may be increased by one by consuming one resource instance. It should be possible for players to use multiple resource instances in one levelling action, thus increasing the level of the weapon by one an equal number of times. A Levelling Manager may be necessary to facilitate the levelling process.
3. Be equipped to a character. As mentioned, once a weapon is equipped, it cannot be equipped to any other character, or more accurately, if the weapon is already equipped, it is removed from the character currently using the weapon before being transferred to the character it is being equipped to.

IV. Resources

Resource instances are used both to level characters and weapons (at least for the first phase of this machine project; phase two would actually see multiple types of resources). One instance is used to level up a character or weapon once. As mentioned, you may need to make a Levelling Manager to facilitate the levelling process. As also previously mentioned, a character and a weapon may be levelled multiple times in one levelling action by specifying the number of resources to be used to level up.

Resources are also used to pull from the Gacha Machine. Each pull costs 300 resources, while a 10 pull costs 2700 resources (a 10% discount). The pulling mechanism is described in greater detail under the section **Gacha and other System Managers**.

Resources are obtained by sending a pair of characters on adventures. Depending on the map the characters are sent and other variables based on the character attributes and weapon attributes, players get a certain amount of resource computed using the following formula:

***Total_Resources** = INT(**Map_Base_Amount** + INT(**Total_Final_Weapon_Power** / 24) * INT(**Total_Character_Influence** / 36) * **Element_Combination_Multiplier**)*

Where:

1. ***Map_Base_Amount*** is the amount of resources a map would normally provide if a pair of characters were deployed on an adventure in them.. See **Appendix E: Map List** for a full list of maps and the amounts of resources they would normally provide.
2. ***Total_Final_Weapon_Power*** is computed as the sum of the final power values of each of the weapons equipped by the two characters deployed on the adventure. The final power value of a weapon is calculated as: **Power of weapon** * **Rarity Multiplier of weapon** + **Level of weapon**
3. ***Total_Character_Influence*** is computed as the sum of the character influence values of each character deployed on the adventure. The character influence value of a character is calculated as: **Character Level** * (1 + ((**Character Rarit**y - 1) / 5))
4. Element_Combination_Multiplier is a rewards multiplier resulting from the element combination of the two characters deployed on an adventure. The value of this multiplier may be found in **Appendix A: Elements and Element Combinations**

V. Enemies

Enemies prevent characters from finishing adventures. A map may have several enemies, and each enemy has (but is not limited to) the following properties:

1. Name - A string that serves as the label for an enemy.
2. Power - An integer that determines the amount of power the enemy will have. The total power of enemies must be overcome by characters in order for adventures to become successful.

See **Appendix D: Enemy List** for a full list of enemies and their attributes.

VI. Maps

Maps are where characters are deployed in order to go on adventures, with each map being the venue of one adventure. Each map has (but is not limited to) the following properties:

1. Name - A string that serves as the label for the map.
2. Base amount - An integer that determines how much resources would be obtained by the player if they were to deploy their characters on an adventure in the map. This value ranges from 50 to 100.
3. Enemy list - A list containing various enemies.

See **Appendix E: Map List** for a full list of maps and their attributes.

In order for an adventure to be successful, the superiority of the characters the player deploys on the map must exceed the superiority of all the enemies in the map. Character superiority is calculated as:

Character_Superiority = ***Total_Final_Weapon_Power*** * (***Total_Character_Influence*** / 10)

See section **IV. Resources** for the formulas used to calculate ***Total_Final_Weapon_Power*** and ***Total_Character_Influence***. ***Enemy_Superiority*** on the other hand is calculated as the **sum of the Powers of all the enemies** in the Enemy list of a map.

Depending on the discrepancy between ***Character_Superiority*** and ***Enemy_Superiority***, the adventure could either be *successfully* completed or *excellently* completed. If ***Character_Superiority*** is greater than ***Enemy_Superiority***, then the adventure is successfully completed, however if the ***Character_Superiority*** is at least 50% more than ***Enemy_Superiority***, then the adventure is excellently completed. Remember that a successfully completed adventure levels the characters involved in the adventure up by 1, while an excellently completed adventure levels the characters involved in the adventure by 2.

VII. Gacha and other System Managers

To facilitate the various processes in the simulator, you need to create a Gacha Machine entity and several other manager entities. The manager entities are not necessary, but they will help in the encapsulation and abstraction of your project (as without them, a lot of the processes mentioned above would have to exist in the Driver or Main class, which would work, would not result in the best design).

For the Gacha Machine, this entity must have behaviour that allows you to:

1. Pull a single character. See **Appendix B: Character List** for a full list of characters. This must also follow the rarity probabilities described in section **II. Characters** (i.e. 50% chance of being 1-rarity, 35% chance of being 2-rarity and 15% chance of being a 3-rarity). Pulling must also cost 300 resources.
2. Pull 10 characters. This behavior performs the Pull a single character behavior 10 times, and must cost 2700 resources (300 less than if you had pulled 10 times manually).
3. Pull a single weapon. See **Appendix C: Weapon List** for a full list of weapons. This must also follow the rarity probabilities described in section **III. Weapons** (i.e. 50% chance of being 1-rarity, 35% chance of being 2-rarity and 15% chance of being a 3-rarity). Pulling must also cost 300 resources.
4. Pull 10 weapons. This behavior performs the Pull a single weapon behavior 10 times, and must cost 2700 resources (300 less than if you had pulled 10 times manually).

Apart from the Gacha Machine, as suggested in the previous chapters, you may also make manager entities for the Refining/Merging actions, Levelling, and Adventuring. Finally, you may also make another entity that would serve as the (text-based) user-interface of your game. This entity would handle UI related logic (i.e. taking user inputs, input validation, moving from menu to menu, etc.) and would interact with the managers and other entities you created, loosely following the Model-View-Controller pattern.

VIII. Machine Project Requirements

For this project, you are required to submit two deliverables:

1. A **UML Diagram**, saved as a PDF, demonstrating your design for this phase of the machine project (for MCO1A)
2. A **Java Project** containing your implementation of the Machine Project, along with any external libraries used, via Canvas (for MCO1B)

Note that external libraries must be approved first by your teacher. It is recommended that you submit at least a week before the deadline, and just submit again before the deadline should you have any updates. Note the following requirements:

1. You are required to design the project using a UML Class Diagram and implement the project using the Java Programing Language
2. The implementation and design will require you to use the OOP paradigm. Procedural use of Java is not allowed except for the Main or Driver class.
3. All files for your machine project should be submitted via CANVAS.
4. Failure to submit on time will merit a 0.0 for the MP.
5. As a backup copy, email your source code to your own my.lasalle account. Your email should be sent before the deadline.
6. You will demonstrate your project on a specified schedule during specified weeks of classes. Being unable to show up during the demo, or being unable to answer convincingly the questions during the demo will merit a grade of 0.0 for the machine project.
7. Any requirement not fully implemented and instruction not followed will merit deductions.
8. This project is to be done by pair. Working in collaboration with other pairs, asking other people's help, and/or copying other people's work are considered as cheating. Cheating is punishable by a grade of 0.0 for the course, and a cheating case will be filed with the Discipline Office.

IX. Rubrics for Assessment

UML Design (70 points)

Descriptor	Expert	Adept	Skilled	Novice	Neophyte
Correctness *(30 points)* Refers to the design considerations made by the group, including proper identification of classes, their properties and behaviours, class relationships, etc.	*(30 points)* All design decisions are correct, classes were correctly identified, with correct properties and behaviors. Appropriate class relationships were also selected, with no minor or major errors	*(24 points)* Most design decisions are correct, classes were correctly identified (with at most one mistake), with correct properties and behaviors (with at most three mistakes). Appropriate class relationships were also selected (up to three minor errors or up to one major error may be present; minor errors include mistakes with association while major errors include mistakes in other relationships).	*(16 points)* Some design decisions are correct, classes were correctly identified (with at most two mistakes), with correct properties and behaviors (with at most five mistakes). Appropriate class relationships were also selected, with major errors (up to five minor errors or up to three major errors may be present; minor errors include mistakes with association while major errors include mistakes in other relationships).	*(8 points)* Design decisions are somewhat correct, classes were correctly identified (with at most five mistakes), with correct properties and behaviors (with at most nine mistakes). Appropriate class relationships were also selected, with major errors (up to nine minor errors or up to five major errors may be present; minor errors include mistakes with association while major errors include mistakes in other relationships).	*(0 points)* Design decisions were generally incorrect.
Cohesion *(15 points)* Refers to how much the properties and behaviours of a class belongs together.	*(15 points)* Classes are highly cohesive, with no obvious mistakes.	*(12 points)* Classes are mostly cohesive, with at most three classes with observable mistakes.	*(8 points)* Classes are cohesive, with at most five classes with observable mistakes.	*(4 points)* Classes are somewhat cohesive, with at most seven classes observable mistakes.	*(0 points)* Classes are not every cohesive with more than seven classes with observable mistakes.
Coupling *(15 points)* Refers to how interdependent classes are on each other. Ideally this should be minimized.	*(15 points)* Classes are not tightly coupled, with as few as three pairs of classes being highly coupled.	*(12 points)* Classes are somewhat tightly coupled, with as few as five pairs of classes being highly coupled.	*(8 points)* Classes are generally not coupled, with as few as seven pairs of classes being highly coupled.	*(4 points)* Classes are somewhat coupled, with as many as nine pairs of classes being highly coupled.	*(0 points)* Classes are coupled, with with more than nine pairs of classes being highly coupled.
Construction (UML) *(5 points)* Refers to the construction of the UML Diagram, including the symbols, figures and arrows used.	*(10 points)* No mistakes were committed with the symbols used for access modifiers (i.e. did not use other symbols other than +, - or #; incorrect symbol use falls under correctness). Shapes used to represent classes are all correct and all arrowheads are correct (except for permissible errors as a result of the diagram builder used).		*(6 points)* Only minor mistakes were committed with the symbols used for access modifiers (i.e. did not use other symbols other than +, - or #; incorrect symbol use falls under correctness). Shapes used to represent classes and all arrowheads mostly are correct with only a few minor errors (except for permissible errors as a result of the diagram builder used).		*(0 points)* Major mistakes were found (more than ten cases of omission is considered a major mistake) with the symbols, shapes and arrowheads used in the UML Diagram, despite permissible mistakes.

Java Implementation

Descriptor	Expert	Adept	Skilled	Novice	Neophyte
Adherence to Design *(20 points)* Refers to how faithful the code is to the submitted UML diagram	*(20 points)* No deviations from the UML design was found in the Java code.	*(15 points)* Up to three deviations from the UML design were found in the Java code.	*(10 points)* Up to five deviations from the UML design were found in the Java code.	*(5 points)* Up to seven deviations from the UML design were found in the Java code.	*(0 points)* More than seven deviations from the UML design were found in the Java code.
Robustness *(20 points)* Refers to the absence of excessive logic or syntax errors in the code. Note that if a syntax error exists, the highest possible rating for this criteria is Novice.	*(20 points)* No runtime or syntax errors were detected. No logic errors detected as well.	*(15 points)* At most three runtime/logic errors were detected. No syntax errors were detected as well.	*(10 points)* At most five runtime/logic errors were detected. No syntax errors were detected as well.	*(5 points)* At most seven runtime/logic errors were detected. At most three syntax errors were detected as well..	*(0 points)* More than seven logical errors were detected, and/or more than three syntax errors were detected.
Readability *(15 points)* Refers to the adherence to proper code conventions, use of descriptive identifiers and use of proper internal documentation.	*(15 points)* Code adheres to proper naming and coding conventions, and consistently follows them. Proper internal documentation is also observed if necessary.		*(7.5 points)* Code generally adheres to proper naming and coding conventions, and consistently follows them, with only up to five detectable mistakes found. Proper internal documentation is also observed if necessary.		*(0 points)* Code does not adhere to proper naming and coding conventions, or inconsistently follows them.
Organization *(15 points)* Refers to the proper use of packages to cluster together related classes.	*(15 points)* Exhibits consistency in the organization used for the Java files of the program. No detectable mistakes were found in how the files were organized.		*(7.5 points)* Exhibits good consistency in the organization used for the Java files of the program. Up to five detectable mistakes were found in how the files were organized.		*(0 points)* Exhibits inconsistency in the organization used for the Java files of the program, with more than five detectable mistakes in how the files were organized.

X. Appendix A: Elements and Element Combinations

As mentioned in the section on adventures, pairs of characters can be sent on adventures and depending on their element combinations, the player would get bonuses and multipliers. Below is a table of possible element combinations. Each pairing has a corresponding letter, check the letter in the list found under the table to determine the effects of the element pairing on the adventure the characters would be deployed under.

	Joker	Trigger	Metal	Cyclone	Luna	Heat
Joker	A	D	D	B	C	C
Trigger	D	A	E	E	B	E
Metal	D	E	A	D	D	B
Cyclone	B	E	D	A	F	F
Luna	C	B	D	F	A	F
Heat	C	E	B	F	F	A

A. **Normal Pairing** - Increases adventure rewards by 10% (i.e. if you were to receive 100 resources from a map, you get 110 instead).

B. **Perfect Pairing** - Increases adventure rewards by 75% (i.e. if you were to receive 100 resources from a map, you get 175 instead).

C. **Nice Pairing** - Increases adventure rewards by 50% (i.e. if you were to receive 100 resources from a map, you get 150 instead).

D. **Decent Pairing** - Increases adventure rewards by 25% (i.e. if you were to receive 100 resources from a map, you get 125 instead).

E. **No Effect** - No increase in adventure rewards.

F. **Bad Pairing** - Decreases adventure rewards by 25% (i.e. if you were to receive 100 resources from a map, you get 75 instead).

Appendix B: Characters List

Name	Rarity	Element
Jekyll	1	**Joker**
Earl Robert	1	**Trigger**
Count d'Artagnan	1	**Metal**
Stede	1	**Cyclone**
Kaguya	1	**Luna**
Van Helmont	1	**Heat**
Gray	2	**Joker**
Bonney	2	**Trigger**
Sir William Marshal	2	**Metal**
Teach	2	**Cyclone**
Jeanne	2	**Luna**
Paracelsus	2	**Heat**
Faust	3	**Joker**
Clyde	3	**Trigger**
Masamune	3	**Metal**
Avery	3	**Cyclone**
Arthur	3	**Luna**
Hermes	3	**Heat**

Appendix C: Weapons List

Name	Rarity	Power
Knife	1	130
Rapier	1	140
Revolver	1	150
Mermaid Tears	1	160
Clarent	1	170
English Longbow	1	180
Circe Staff	2	150
Vorpal sword	2	160
Merlin’s Staff	2	170
Five-cross Sword	2	180
Bashosen	2	190
Golden Cudgel	2	200
Philosopher’s stone	3	180
Magic Bullets	3	190
Fragarach	3	200
Honjo Masamune	3	210
Excalibur	3	220
Scythe of Father Time	3	230

Appendix D: Enemies List

Name	Power
Slime	73
Orc	84
Familiar	144
Faerie	175
Elf	224
Sorcerer	313
Hydra	360
Basilisk	499
Harpy	639
Loki	740

Appendix E: Maps List

Name	Base Amount	Enemy List
Underground Caverns	53	1x Elf, 6x Slime
Forest of Enchantments	77	5x Slime, 5x Orc, 3x Familiar, 3x Faerie, 2x Elf, 1x Sorcerer
Sea of Hope	85	75x Slime, 20x Sorcerer, 5x Hydra
Tower of Ether	91	20x Basilisk, 7x Harpy, 5x Loki
Celestial Plane	100	50x Faerie, 20x Hydra, 10x Loki