Assignment 4 - Part A: Sum-Awesome Game

- See website for due date and late penalty info.
- **PART A** is to be done **individually or in pairs**. Do not show other students your code, do not copy code from another person/online. Do not reuse your previous work (even **if retaking the course**).
- **PART B** is to be done **individually**.
- You may use general ideas you find online and from others, but your solution must be your own.
- See the marking guide for details on how each part will be marked.

1. Game Overview

Create a game where the user (player) does simple addition to attack their opponent and win epic equipment!

The game is played over a series of matches. When the player wins or loses a match, they go on to the next match. In each match, the player has a single character, and the AI opponent has three characters.

The game board is a 3x3 grid of integers between 0 and 15 inclusive (possibly with repetition). Each move (or turn) the player picks one of the eight numbers from the outside of the game board. In their head, they add this chosen number to the number in the middle of the board and type the sum into the game to successfully complete their turn.

10	1	5
9	5	7
15	13	10

Example game board

The cell containing the chosen number becomes part of what is called the current "*fill*". Once all eight cells around the edge of the board are part of the fill, it triggers the user's character to attack the opponent(s).

1.1 Turn Details

Each turn the user must type in a number. If the entered number is the sum of the number in the centre cell and the number in one of the outer cells, then it is a successful move. If more than one of the numbers in the outer cells match the entered sum, then:

1. Choose one of the matching cells which is not yet part of this fill.
2. If all matching cells are already part of the fill, then random choose one of them.

A fill has a strength number. The strength is how much base damage this fill's attack will do. Strength initializes to 0 at the start of the fill. As the user enters sums that match with numbers around the outside of the grid, each matched number is then added to the strength of the fill. After a fill is completed and the attack has been resolved, the strength resets to 0 for the next fill.

On a successful turn:

- The game tracks that the matching cell has become part of the fill.
- The number in the matching cell is added to the strength of the fill.
- The number in the middle cell is replaced by the number in the matching cell.
- The number in the matching cell is replaced with random number from the allowed range.

If the user's move was not successful (i.e., it is not the sum of the middle number and any of the outer numbers) then it is a failed move. In this case one opponent character attacks the player character.

Additionally, every 3-5 turns (randomly) one opponent character attacks.

When the user has added all cells around the outside of the board to the fill, it triggers an attack by the player character.

When all characters on the opponents team are defeated the player wins a random equipment reward and a new match begins.

1.2 Attacks

When the user completes a fill, the player character will attack the opponent team. The target of the attack is chosen based on the final cell added to the fill. If the final cell added to the fill was...

- one of the left three cells, then target the left character;
- one of the middle two cells, then target the middle character;
- one of the right three cells, then target the right character.

Targeting can be modified by the player character's weapon.

When the opponent attacks the they will do damage to the player character.

The amount of health that each character has, and the amount of damage that enemy characters do, is unspecified. You should pick a number that allows the user to win the game if playing well, but lose the game if playing poorly.

1.3 Weapons

The player character can have zero or one weapons. At the start of the game, the character has no weapons but might earn one when they win a match.

Each weapon has a name and boosts the user's attack in certain ways based on some property. The weapons are:

1. **Lightning Wand**: If the user completes a fill in less than 10s (from the first sum they enter until the fill is complete), the Lightning Wand will target an additional random opponent character for 100% damage. The additional attack will not target dead characters, and might target the same character as the normal attack.
2. **Fire Staff**: If the user completes a fill with 15 or more cells (by re-selecting cells which are already part of the fill, but have new numbers) then the attack hits the chosen opponent for 100% damage, and the opponents to either side (if any) for 50% damage. If the primary target is already dead, the 50% damage still applies to the side targets.
3. **Frost Bow**: If the player selects numbers for the fill in ascending order, then the attack hits all opponents for 100% damage.
4. **Stone Hammer**: If the user completes a fill with 10 or more cells, then the attack hits all opponents at 80% each.
5. **Diamond Sword**: If the user selects numbers for the fill in descending order, then the attack

hits the main target for 100% and opponents to either side for 75%.

6. **Sparkle Dagger**: If the user completes a fill in less than 20s then the attack targets an additional random opponent character for 50% damage (similar to Lightning Wand).

1.4 Rings

The player character may have up to three (3) rings equipped. There are numerous types of rings in the game. When the user completes a fill, based on certain properties of the fill, a ring may do an additional effect. Let `S` be the strength of the fill. Your game must support at least the following effects:

1. **The Big One**: 50% damage bonus if `S >= 160`
2. **The Little One**: 50% damage bonus if `S <= 90`
3. **Ring of Ten-acity**: 50% damage bonus if `S % 10 == 0`
4. **Ring of Meh**: 10% damage bonus if `S % 5 == 0`
5. **The Prime Directive**: 100% damage bonus if `S` is prime
6. **The Two Ring**: 1000% damage bonus if `S` is a power of 2

Multiple rings may activate on one fill. You choose how you want the bonuses to combine:

- *Multiply* them: a 10% and 50% bonus could give a combined 65% bonus damage (1.10 * 1.50).
- *Add* them: a 10% and 50% bonus could give a combined 60% bonus damage.

You may adjust these damage bonuses or thresholds to balance your game, but you must have at least 3 different amounts of damage bonus.

Damage should be rounded to the nearest integer.

2. Output

2.1 Before Each Move

Before the user enters their move (the sum), they must be shown:

- Health of opponent characters in each location.
- Game board of numbers, including which are part of the current fill already.
- Total strength of the current fill.

You can customize the look of the output. Here is a possible game board display:

```
     [400]    [500]    [200]

       4        5       _10_

       6        8        11

       0       _4_       15

             [ 620 ]       Fill Strength: 139
Enter a sum:
```

- Across the top are the health of the three opponent characters.
- The underlined numbers are the cells which are part of the current fill.
- The user character's health is shown at the bottom.

2.2 Attack Output

During a player's attack, the game must display to the user text explaining how gear affects the attack. For example, the output could look like:

```
Enter a sum: 14

Fill complete! Strength is 145.
The Big One adds 50% bonus damage.
Ring of Ten-acity adds 50% bonus damage.
Hit left character for 290 damage.
Lightning Wand targets right character.
Hit right character for 290 damage.
Kills right character!
```

The output format is flexible, but the player must be able to see what modifiers were applied, and how much damage was done to which characters, and if an enemy character was killed.

If an attack targets a character who is already dead (either because the user finished the fill with a cell that lined up with the dead character, or because a weapon caused that space to be targeted), the output should state that the attack missed. For example:

```
Fill strength 131.
Missed left character.
```

or if targeting the middle character but both left and middle characters are dead:

```
Fill strength 131.
Misses middle character.
Diamond Sword targets left character.
Missed left character.
Diamond Sword targets right character.
Hits right character for 98 damage.
```

2.3 User Input

Each turn (or move) the user enters a command. Here are the possible commands:

- **`##`:** (An integer) Check if this value matches the sum of the centre number and another number and handle it as a success or failure.
- **`gear`**: display the name and ability of the player character's current gear:
  - Weapon (if any)
  - Rings (if any)
- **`cheat <command>`**: where `command` is one of the following:
  - **`lowhealth`**: All enemies in this match are made to have very low health (takes ~1 hit to kill). This setting persists only until the end of the match.
  - **`highhealth`**: All enemies in this match are made to have high health (hard to kill). This setting persists only until the end of the match.
  - **`weapon #`**: The player equips the chosen weapon and keeps it for future matches too. "weapon 0"

  removes any weapons. If the number is not a valid weapon number then display an error message.
  - **`rings # # #`**: The player equips the three rings listed by number and keeps them for future matches. This replaces the user's current three rings. A number may be 0, in which case it means no ring. It does not matter the order in which the rings are held (so "cheat rings 1 2 0" is the same as "cheat rings 0 1 2").
  - **`max #`**: All new values coming into the grid must be between 0 and the # inclusive. Do not change current grid. Display an error message if # is less than 1. This setting persists only until the end of the match.
- **`stats`**: Display stats for:
  - How often each ring and each weapon has activated.
  - Number of matches won and lost.
  - Total amount of damage the player has done over all matches.
  - Total amount of damage the player has received over all matches.
  - Number of fills completed.
- **`new`**: the current match exits and counts as a loss, and a new one starts.

Otherwise, display an error message and ask the user to retry.

2.4 Stats

You must track a number of events during game play. Statistics are not saved between game executions. The display must line up the numbers nicely (right-justified).

Possible stats display might look like:

```
Equipment Activations
    Lightning Wand          10
    Frost Bow                5
    The Little One         101
    Ring of Meh              1
Matches:
    Won                      9
    Lost                     4
Total Damage
    Done                10,402
    Received             1,523
Fills Completed            401
```

2.5 Winning / Losing

If all opponent characters are reduced to 0 health (character health cannot be negative), then the player wins. The player is then given a random new item which may be either a weapon or a ring. It may be an item they already have, or a different one. The player is then asked if they would like to equip this item. The player may have at most 1 weapon, and at most 3 rings equipped. If the user wants to equip a ring but have no available slots, they must be asked which ring to replace. The player may equip multiple copies of the same ring if given a second copy.

Replaced items and unused new items are lost (not kept in an inventory).

3. Implementation Requirements

Packages

- Your program must be separated into at least two package: one for the UI, one for the model.
- Your game model package must never print to the screen. The model must implement the Observer pattern:
    - The game model provides an interface to register for notifications of events.
    - The UI registers an observer (which is an object that implements this interface) with the game model.
    - The UI's observer then takes care of displaying events to the user.
    - The model's observer should not emit text (like "The Big One adds 50% damage bonus."), but rather emit an object which has structure, such as identifies the source (a string?), what happened (an enum?), and information about the event. See stats implementation below.
- Your game model package must never read from the keyboard.

Avoid Null

- Use the Null Object pattern to represent when the user has no weapon.
- You must *try* to reduce the number of occurrences of the keyword `null` in your code. Use either Null Object pattern, or `Optional` when reasonable.

Flexible Design

- You must follow the Open-Closed Principle. **This is important! This was a central idea in designing this assignment! It is what you are expected to learn.**
- You must *not* have a single weapon class or ring class that implements *all* of the code for that item.
- You must favour composition over inheritance.
- For example, you must be able to add a new ring or a new weapon without changing existing code (other than where that item is created). It should be easy to combine the behaviours of existing weapons or rings in new ways.

3.1 Stats Implementation

There must be one (or more) classes dedicated to tracking the stats: tracking stats must not be distributed throughout the code.

You must use the Observer pattern for stats. The observer pattern is already required by the assignment to avoid model outputting to the screen. Your stats class(es) will use the same mechanism to register an observer with the game model. Your stats class(es) will then receive notifications of game events which it will track.

The count of equipment activations must persist while playing the game, even if that equipment is later replaced. For example, if the player has a Fire Staff and it activates twice before being replaced by the Frost Bow, then the two activations of the Fire Staff must still be displayed in the statistics. If the user later gets a new Fire Staff then it should resume adding to the same tally as before. The same applies to rings.

Hint: you may want to use a hash-map to track how many times a piece of equipment has activated.

4. Deliverables

Submit a ZIP file of your project to CourSys. See course website for directions on creating and testing your ZIP file for submission. All submissions will automatically be compared for cheating.

5. Version History

1. Initial release