German University in Cairo
Media Engineering and Technology
Prof. Dr. Slim Abdennadher

Computer Programming Lab, *Spring 2022*
Marvel - Ultimate War: **Milestone 2**

Deadline: 19th of May 2022, 11:59 PM

This milestone is a further *exercise* on the concepts of **object oriented programming (OOP)**. By the end of this milestone, you should have a working game engine with all its logic, that can be played on the console if needed. The following sections describe the requirements of the milestone. Refer to the **Game Description Document** for more details about the rules.

- Please note that you are not allowed to change the hierarchy provided in milestone 1 nor the visibility/access modifiers of variables except as explicitly stated in this milestone description. You should also conform to the method signatures provided in this milestone. However, you are free to add more helper methods. You should implement helper methods to include repetitive pieces of code.
- All methods mentioned in the document should be `public`. All class attributes should be `private` with the appropriate access modifiers and naming conventions for the getters and setters as needed.
- You should always adhere to the OOP features when possible. For example, always use the `super` method or constructor in subclasses when possible.
- The model answer for M1 is available on CMS. It is recommended to use this version. A full grade in M1 doesn't guarantee a 100 percent correct code, it just indicates that you completed all the requirements successfully :)
- Some methods in this milestone depend on other methods. If these methods are not implemented or not working properly, this will affect the functionality and the grade of any method that depends on them.
- **You need to carefully read the entire document to get an overview of the game flow as well as the milestone deliverables, before starting the implementation.**

1 Interfaces

1.1 Damageable Interface

Name : `Damageable`

Package : `model.world`

Type : Interface

Description : Interface containing the methods available to objects on the map that can take damage. All `Champion`s and `Cover`s are `Damageable`s.

You should add the following methods to this interface:

1. `Point getLocation()`: A method that returns the location of the Damageable object on the board.
2. `int getCurrentHP()`: A method that returns the current health points remaining for the Damageable object.
3. `void setCurrentHP(int hp)`: A method that updates the current health points remaining for the Damageable object.

2 Exceptions

2.1 Brief description of the previously implemented exceptions

1. `NotEnoughResourcesException`: Is thrown when trying to perform any action without having enough resource(s) for this action.
2. `LeaderAbilityAlreadyUsedException`: Is thrown when the leader champion of any player tries to use his leader ability after it has already been used before. Recall that every leader can only use his leader ability once per game.
3. `UnallowedMovementException`: Is thrown when a champion is trying to move while violating the move regulations.
4. `AbilityUseException`: Is thrown when a champion is trying to cast an ability on an out of range target or being in a condition that prevents him from casting an ability.

2.1.1 GameActionException

The following modification should be done: No objects of type `GameActionException` can be instantiated.

Further, the following **new** exceptions should be implemented:

2.2 LeaderNotCurrentException

Name : `LeaderNotCurrentException`

Package : `exceptions`

Type : Class

Description : A subclass of `GameActionException` representing an exception that is thrown upon trying to use the leader's ability while the champion whose turn is running is not a leader.

2.2.1 Constructors

1. **LeaderNotCurrentException()**: Initializes an instance of a `LeaderNotCurrentException` by calling the constructor of the super class.
2. **LeaderNotCurrentException(String s)**: Initializes an instance of a `LeaderNotCurrentException` by calling the constructor of the super class with the input message.

2.3 InvalidTargetException

Name : `InvalidTargetException`

Package : `exceptions`

Type : Class

Description : A subclass of `GameActionException` representing an exception that is thrown when a champion tries to cast an ability on an invalid target or an empty cell.

2.3.1 Constructors

1. **InvalidTargetException()**: Initializes an instance of a `InvalidTargetException` by calling the constructor of the super class.
2. **InvalidTargetException(String s)**: Initializes an instance of a `InvalidTargetException` by calling the constructor of the super class with the input message.

2.4 ChampionDisarmedException

Name : `ChampionDisarmedException`

Package : `exceptions`

Type : Class

Description : A subclass of `GameActionException` representing an exception that is thrown when a champion tries to use a normal attack while being disarmed.

2.4.1 Constructors

1. **ChampionDisarmedException()**: Initializes an instance of a `ChampionDisarmedException` by calling the constructor of the super class.
2. **ChampionDisarmedException(String s)**: Initializes an instance of `ChampionDisarmed Exception` by calling the constructor of the super class with the input message.

3 Effects

3.1 `Effect` Class

No objects of type `Effect` can be instantiated. This class should implement the `Cloneable` interface (predefined in Java). Implementing this interface will help us create an exact copy of any subclass of `Effect` without needing to call the constructor of its corresponding class. When overriding the needed method from the `Cloneable` interface, you should override it as a `public` method.

This class should also include the following additional methods:-

- `void apply(Champion c)`: This method applies the logic of the effect on the given champion (Based on what is illustrated in the below table).
- `void remove(Champion c)`: This method should undo the logic that the `apply` method previously did on the given Champion. This method is called once the duration of the effect is over.

NOTES:
1- If the logic applied by the effect contains permanent parts, these parts will not be undone when calling the remove method.
2- A champion can carry multiple instances of the same effect. Think how this can affect the `remove` method.

Name	Effect
Disarm	Target cannot use normal attacks. Gain a SINGLETARGET damaging ability called "Punch" costs: 0, damage: 50, cooldown: 1, range: 1, actions: 1
Dodge	Target has a 50% chance of dodging normal attacks. Increase speed by 5%.
Embrace	**Permanently** add 20% from maxHP to currentHP, **Permanently** increase mana by 20%, Increase speed and attackDamage by 20%.
PowerUp	Increase damageAmount and healAmount of all damaging and healing abilities of the target by 20%.
Root	Target cannot move.
Shield	Block the next attack or damaging ability cast on target. Once an attack or ability is blocked, the effect should be removed. Increase speed by 2%.
Shock	Decrease target speed by 10% Decrease the target's normal attack damage by 10% Decrease max action points per turn and current action points by 1.
Silence	Target cannot use abilities. Increase max action points per turn and current action points by 2.
SpeedUp	Increase speed by 15%. Increase max action points per turn and current action points by 1.
Stun	Set target to INACTIVE. Target is not allowed to play their turn for the duration.

4 Abilities

4.1 `Ability` Class

No objects of type `Ability` can be instantiated.

This class should also include the following additional methods:-

- `void execute(ArrayList<Damageable> targets)`: This method should handle applying the logic of any ability on every Damageable target in the given array list. Every subclass should implement it appropriately.
  Note: The `execute` method does not need to validate the given targets. The targets will be validated before calling the method. Therefore, the given targets to the method should always be considered as valid targets

5 World

5.1 `Champion` Class

No objects of type `Champion` can be instantiated. This class should implement the `Comparable` interface: The comparable interface is a predefined one in Java, and enables objects that implement it to be compared against each other. In our case, this will help us determine the turn order of the champions based on the speed attribute. The higher the speed of the champion, the sooner he will get to use his actions. In case the speeds of the compared champions are identical, you should resort to the lexicographic order of the champions' names.

This class should also include the following additional method:-

- `void useLeaderAbility(ArrayList<Champion> targets)`: This method is called whenever a player chooses to use his leader ability. It should be handled differently for each corresponding leader type and should apply the leader ability's logic on the `targets` arraylist.

  The following describes the leader ability of each type:-
  - **Hero:** Removes all negative effects from the player's entire team and adds an Embrace effect to them which lasts for 2 turns.
  - **Villain:** Immediately eliminates (knocks out) all enemy champions with less than 30% health points.
  - **AntiHero:** All champions on the board except for the leaders of each team will be stunned for 2 turns.

6 Engine

6.1 `Game` Class

Important notes:

- Upon creating a new `Game` instance, the turn order of both player's champions should be prepared.
- For all methods that represent the possible actions the champion can take in their turn, you are required to validate that action before carrying out the action's logic. For example: a champion cannot move to a non empty cell or cast an ability if he does not have the necessary mana or action points, etc.
- For all abilities that may affect more than one target, you should note the following:-
  - Healing abilities can only affect friendly champions.
  - Damaging abilities can only affect enemy champions and covers.
  - Crowd control abilities that positively affect the targets will only affect friendly targets. The ones that negatively affect the targets will only affect enemy targets.

  For normal attacks or abilities that target only one target, you should enforce the same rules and prevent the use of the ability or attack entirely if any of the rules is violated.
- Champions can carry multiple effects that influence their condition between ACTIVE, INACTIVE, and ROOTED. When a new effect is applied or removed from a champion, you should take into consideration in which condition the champion will be. Note: INACTIVE has a priority over ROOTED. For example, if a champion is INACTIVE then a Root effect is applied to him, his condition will remain INACTIVE.
- Necessary exceptions should be thrown when needed.
- If any action done by a champion resulted in the death of some champion(s), that champion(s) should immediately be removed from any place that they existed in.
- Whenever a distance between two champions is needed to be calculated, Manhattan distance calculation is always used.

- For all cast ability methods, the method should only pass every possible valid target to the `execute` method of the ability based on the ability's type, area of effect and cast range.
- For the attack method and all cast ability methods (except for casting a single target ability), if no valid target was found based on the area of effect and the cast range of the ability, no one will be affected by the action. However, all required resources for this action will be deducted from the champion carrying it out.
- Abilities with SURROUND area of effect can only affect specific points (if possible) on the board regardless of the cast range of the ability. These points are shown in the below figure. If we assume that the champion casting the ability is located at the yellow cell, then the points that the ability can affect are the blue cells.

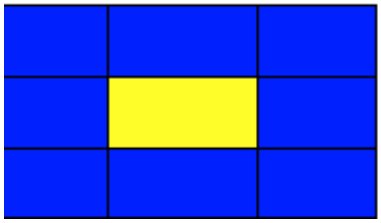

Figure 1: Possible points that surround abilities can affect

This class should also include the following additional methods:-

1. `public Champion getCurrentChampion()`: This method merely returns the champion who's current turn is taking place.
2. `public Player checkGameOver()`: This method checks whether the game is over or not. If it is, it should return the winning player otherwise, it should return null.
3. `public void move(Direction d)`: A method that is called when the current champion wishes to move in a particular direction. A move requires one action point in order to be performed.
4. `public void attack(Direction d)`: A method that is called when the current champion wishes to perform a normal attack in a particular direction. The method should retrieve the first Damageable within the champion's normal attack range at the given direction and perform the attack on it.

   Carefully consider the special interaction between the different champion types as well as the different effects that the target can have. Refer to the game description for that interaction. Champions should deal 50% extra damage if the interaction condition is met. All normal attacks require two action points in order to be performed.
5. `public void castAbility(Ability a)`: A method that is called when the current champion wishes to cast an ability that is not limited to a direction or a particular target.
6. `public void castAbility(Ability a, Direction d)`: A method that is called when the current champion wishes to cast an ability with DIRECTIONAL area of effect.
7. `public void castAbility(Ability a, int x, int y)`: A method that is called when the current champion wishes to cast an ability with SINGLETARGET area of effect and that target is located at cell (x,y) on the board.
8. `public void useLeaderAbility()`: A method that is called when the player wishes to use their leader's ability, this method should make sure that the current champion whose turn is taking place is actually a leader. This method should pass the correct target champions to the `useLeaderAbility` method declared in that Champion's class. The choice of which targets are to be passed is based on the type of the leader champion casting the ability. For example, if the leader casting the ability is a Hero champion, then only that champion's team should be passed to the `useLeaderAbility` method of that champion.

9. `public void endTurn()`: This method is called when the current champion decides to end his turn. The method will then remove that champion from the turn order queue. Then, If the turn order queue is empty after removing the current champion, the turns of the champions should be prepared. However, if the turn goes to an INACTIVE champion, the method should skip this champions' turn. The new current champion is therefore the first non INACTIVE champion in the queue. The method should then prepare the turn of the new current champion by having his effects and ability timers updated. Also, his current action points should be properly reset.
10. `private void prepareChampionTurns()`: This method adds all non dead champions of both players to the turn order queue.