Lab 2: Inheritance

1. Instruction

1.1. Click the provided link on CourseVille to create your own online submission repository.

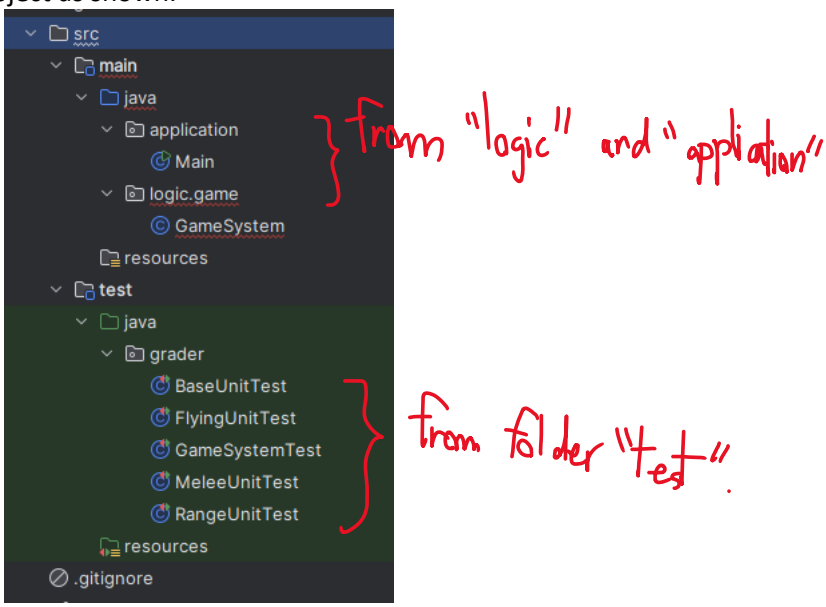
1.2. Open IntelliJ and then create a gradle project and set project name in this format **progMethLab02**

1.3. **Use build.gradle, settings.gradle, and .gitignore provided in the assignment zip file in MCV. Change the name of the main class and project name as appropriate.**

1.4. Initialize git in your project directory (if IntelliJ has not done it for you).

1.4.1. Commit and push initial codes to your GitHub repository.

1.5. Put the files into your project as shown:



1.6. Implement all the classes and methods following the details given in the problem statement file which you can download from CourseVille.

1.7. Test your codes with the provided JUnit test cases, they are inside folder **test/grader (copy it to the correct gradle test folder, as shown in the picture).**

1.7.1. If you want to create your own test cases, please put them inside package **test.java.student (in your gradle test folder)**

1.7.2. Aside from passing all test cases, your program must be able to run properly without any runtime errors.

1.8. After finishing the program, create a UML diagram for classes in package logic.unit and logic.game and put the result image (YOU MUST EXPLICITLY CREATE UML.png file) in project folder.

1.9. Create a jar file using gradle.

1.10. Push all commits to your online submission GitHub repository.

2. Problem Statement: Zerg chess

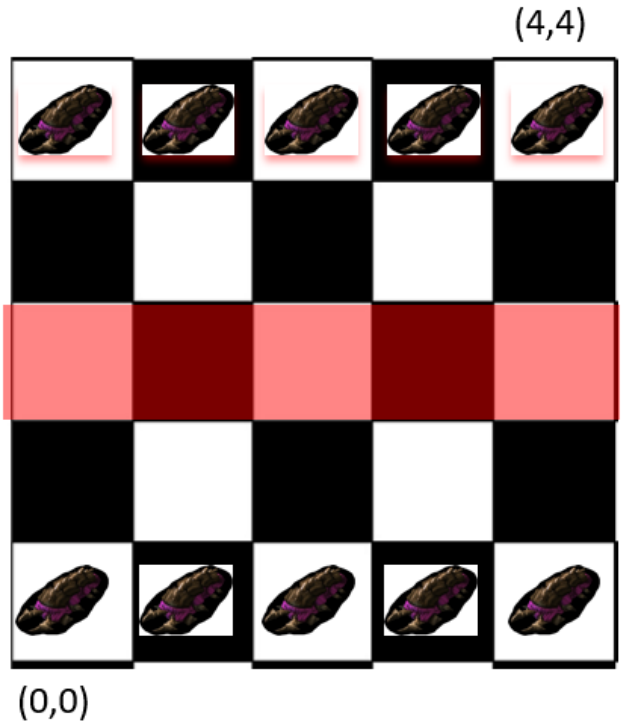


Figure 1 Zerg chess

Zerg chess is a fairy chess, chess with the special rules, which is popular among Prog-meth TAs. We have implemented this chess in text game version, but we have no time to finish it, so we want you to help us implement the game.

The rules are less complex than traditional chess. Win condition is attacking opponent's pieces until there are 2 or less pieces on the opponent's side. The game starts with 10 base units along 2 sides of the players. Units can be in the same tile for this game.

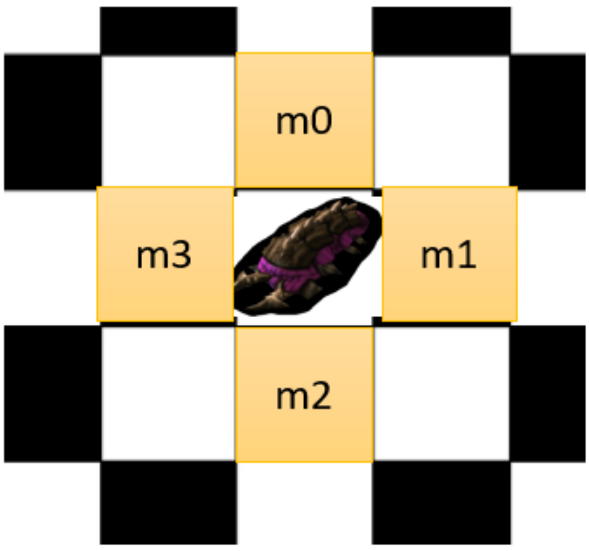


Figure 2 Base unit move patterns

Base units can move in just 4 directions like in Figure 2. Base units have 2 HP and power 1. When they move to the tile if there are any opponent's units on that tile, base units will attack all of them by their power.

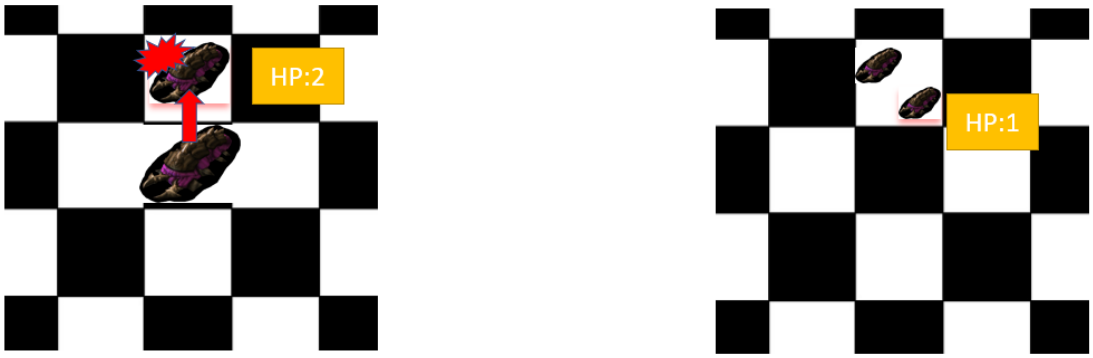


Figure 3 attacking

When a piece's HP is below 0, it will be removed from the game.

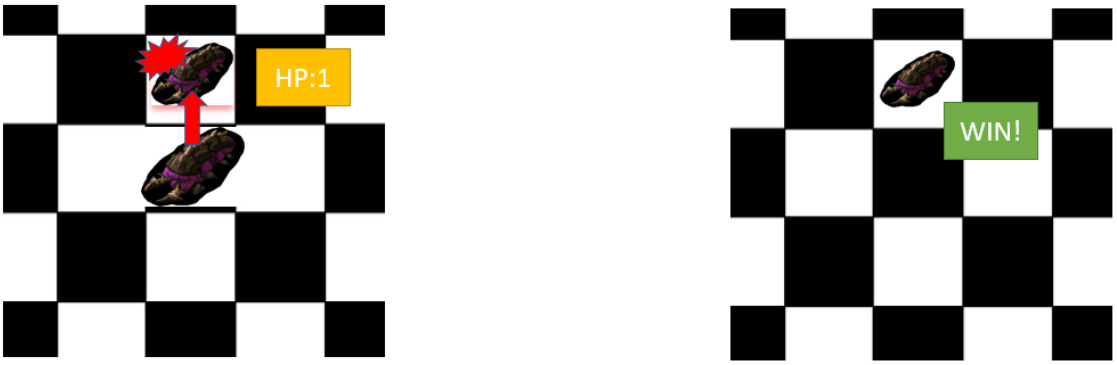


Figure 4 removing dead piece

The center row of the board (highlighted red in Figure 1 and Figure 5) is the promote line. When a unit enters this line, it will be promoted. There are 3 classes of units which units can be promoted to; Melee unit, Flying unit, and Range unit. When the unit is promoted, its HP will be fully restored.

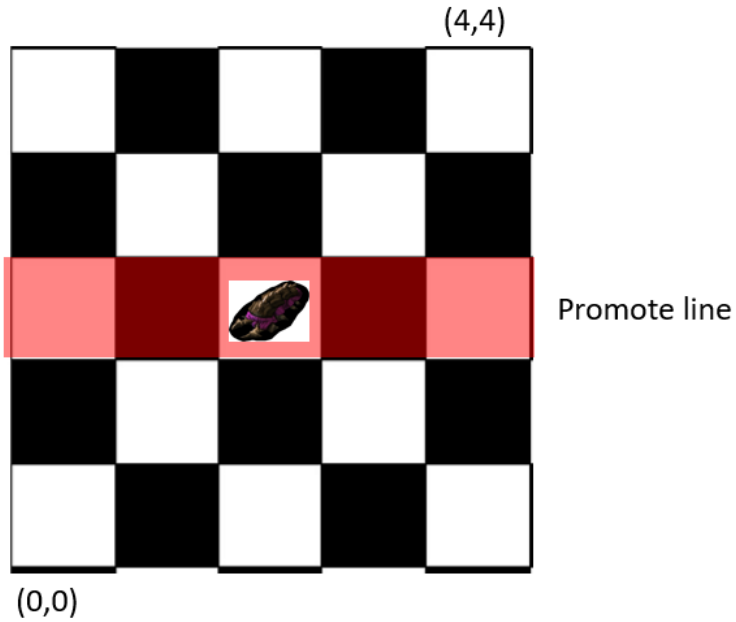


Figure 5 Promote line

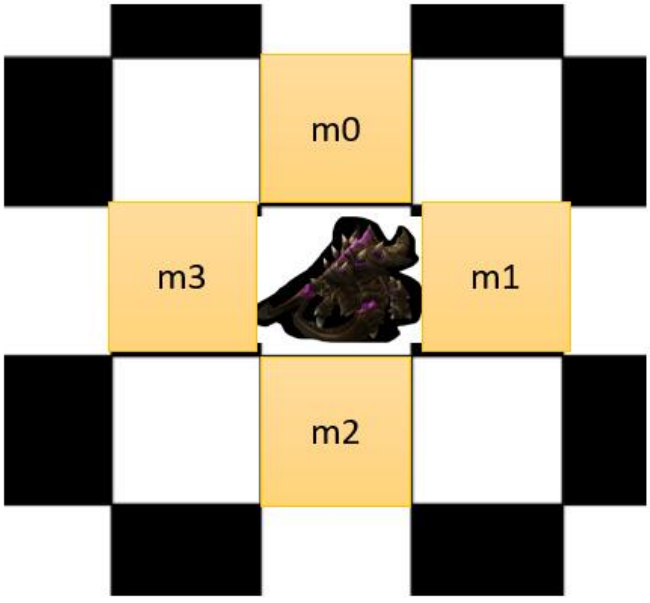


Figure 6 Melee unit move patterns

Melee unit can move in 4 directions and attack like base units. However, its HP is 5 and power is 2.

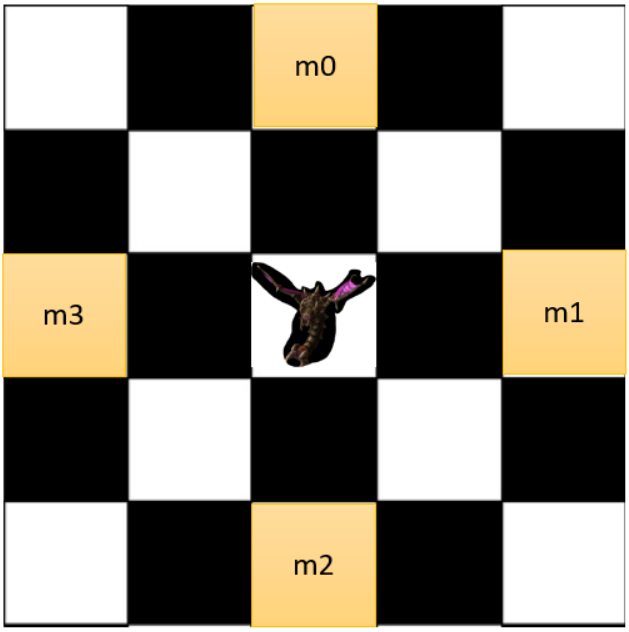


Figure 7 Flying unit move patterns

Flying unit's HP, power and attack rule is same as base unit, but move patterns are further for 1 tile like in figure 7. Another unique attribute of flying units is that base unit and melee unit cannot attack them because they are flying. Even flying units themselves cannot attack other flying units.

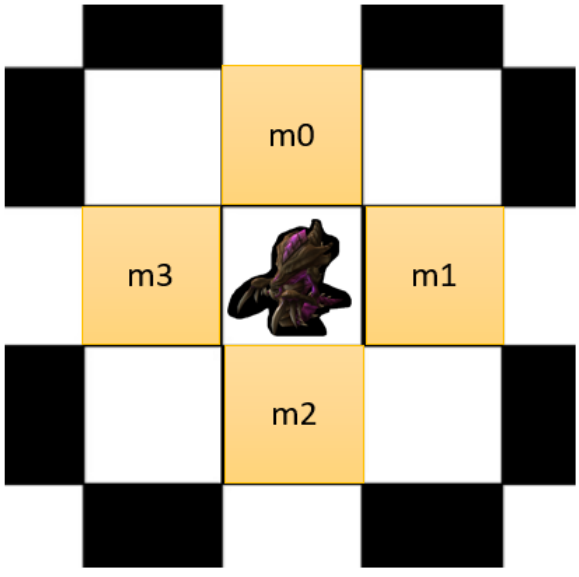


Figure 8 Range unit move patterns

The last type of unit is range unit. Range unit's hp and power is same as base unit. Its move patterns are the same as base and melee unit, but they attack the tile in front of them after their move. The attack directions depend on side of the unit (we have White unit

on one side and Red unit on the other side). Range unit is the only unit type that can attack a flying unit.

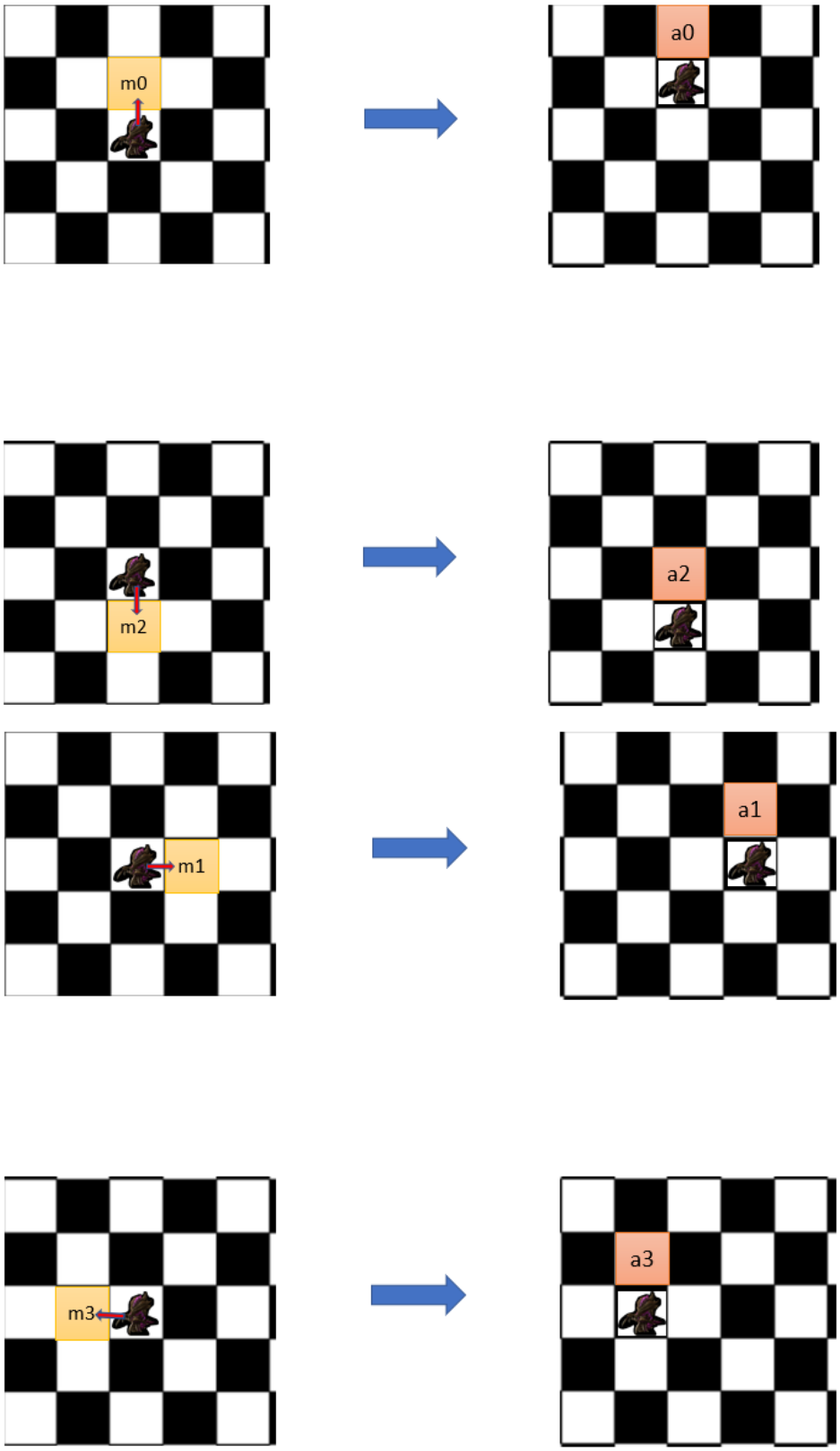


Figure 9 White range unit attack example

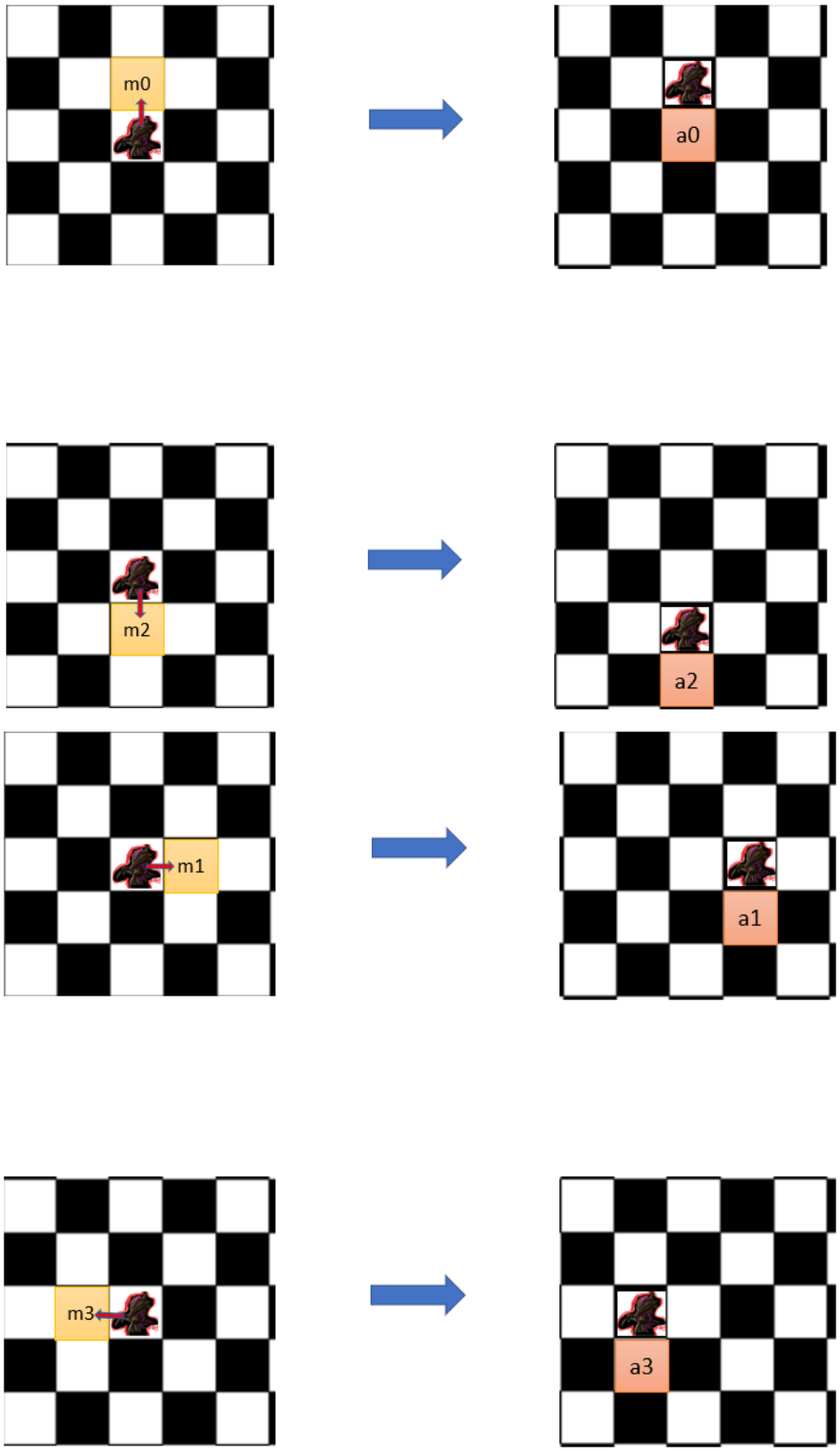


Figure 10 Red range unit attack example

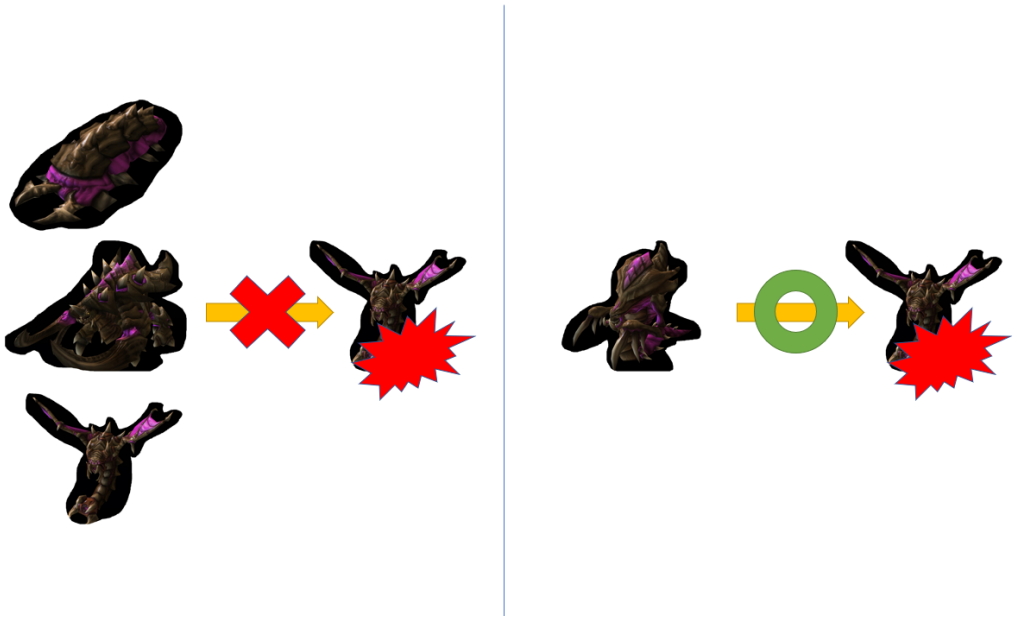


Figure 11 Attacking rules

3. Implementation Detail

The class package is summarized below.

*** In the following class description, only details of IMPORTANT fields and methods are given. ***

3.1 Package logic.unit /* You must implement this package from scratch */

3.1.1 class BaseUnit /* You must implement this class from scratch */

This class is a basic type of units. It contains many common elements which a unit should have. The base unit's hp is 2 and power is 1.

Field

Name	Description
- int row	Row which this unit is on
- int column	Column which this unit is on
- boolean isWhite	True if the unit is on the white side. False if the unit is on the red side.
- String name	Name of the unit which will be displayed and used to identify the unit.
# int hp	The unit's hp

# int power	The unit's power
# boolean isFlying	True if the unit is flying. False if the unit is not flying.

Method

Name	Description
+ BaseUnit(int startColumn, int startRow, boolean isWhite, String name)	Assign power as 1 and isFlying as false because these 2 fields will not be changed. Set hp as 2 Set column and row as startColumn and startRow Set other fields value as in the parameter.
+ boolean move(int direction)	Check if the unit will be able to move in the direction in the parameter. Moving directions are all in the same way in both red and white units Direction 0: go up 1 tile (row + 1) Direction 1: go right 1 tile (column + 1) Direction 2: go down 1 tile (row - 1) Direction 3: go left 1 tile (column - 1) If the unit can move in the direction, move it and return true. If there is any other case which does not meet the conditions (direction is not 0-3/border), return false.
+ void attack(ArrayList<BaseUnit> targetPieces)	Check all units in targetPieces, if there are any units that are not flying and in the same tile as this unit, attack them. Print out **this.getName() + " attacks " + [that unit].getName()** and decrease those units' hp by this unit power.
+ getter and setter of all fields	Setter for isFlying and power are not necessary.

	If the number when setting row or column is below 0, set the actual value to 0. If the number when setting row or column is more than 4, set the actual value to 4.

3.1.2 class MeleeUnit **/* You must implement this class from scratch */**

This class is a class for melee type units. Almost all rules of this unit are same as base unit but this unit's hp is 5 and power is 2.

Method

Name	Description
+ MeleeUnit(int startColumn,int startRow, boolean isWhite, String name)	This is the Constructor. Almost all settings are the same as base unit (Use super for less code) Set this unit's hp as 5 and power as 2

3.1.3 class FlyingUnit **/* You must implement this class from scratch */**

This class is a class for flying type units. Flying units have a unique move rule.

Method

Name	Description
+ FlyingUnit(int startColumn,int startRow, boolean isWhite, String name)	This is the Constructor. Almost all settings are the same as base units (Use super for less code) Set this unit's hp as 2 (to be fully restored) Set isFlying as true
+ boolean move(int direction)	Move patterns are similar to the one of base units. Check if the unit will be able to move in the direction in the parameter. Moving directions are all in the same way in both red and white units

	Direction 0: go up 2 tile (row + 2) Direction 1: go right 2 tile (column + 2) Direction 2: go down 2 tile (row - 2) Direction 3: go left 2 tile (column - 2) If the unit can move to the direction, move it and return true. If there is any other case which does not meet the conditions (direction is not 0-3/border), return false.

3.1.4. class RangeUnit /* You must implement this class from scratch */

This class is a class for range type units. Range units have a unique attack rule.

Method

Name	Description
+ RangeUnit(int startColumn,int startRow, boolean isWhite, String name)	This is the Constructor. Almost all settings are the same as base unit (Use super for less code) Set this unit's hp as 2 (to be fully restored)
+ void attack(ArrayList<BaseUnit> targetPieces)	Check all units in targetPieces. If this unit is white, it will attack on a tile in a row above this unit. If this unit is red, it will attack on a tile in a row below this unit. Print out **this.getName() + " attacks " + [that unit].getName()** and decrease target units' hp by this unit power.

4.2. package logic.game /* You must implement something in this package */

4.2.1. class GameSystem/* You must implement 1 method in this class */

This class is the main game system which is singleton. This class provide board status and game rules. Game will play via this class.

Fields

Name	Description
- ArrayList<BaseUnit> allWhitePieces	ArrayList containing all white pieces in the game
- ArrayList<BaseUnit> allRedPieces	ArrayList containing all red pieces in the game
- boolean gameEnd	True if the game ended
- GameSystem instance	A static instance of game system this will make sure that there is only one game system when program is running.

Method

Name	Description
- GameSystem()	This is the Constructor. Initialize the Piece ArrayLists and all pieces. Add all the pieces to its ArrayList. Set gameEnd to false.
+ void printBoardStatus(ArrayList<BaseUnit> allPieces)	Print all pieces' names and position in allPieces
+ boolean removeDeadPieces(ArrayList<BaseUnit> allPieces)	Remove all pieces with 0 or below hp in allPieces
+ BaseUnit promote(BaseUnit baseUnit,int choice)	/* FILL CODES */ This method is for promoting unit Return a new unit whose type follows the choice in parameter and whose fields follows baseUnit's parameter.

	Choice 0: Melee unit Choice 1: Range unit Choice 2: Flying unit
getter/setter for gameEnd, allWhitePieces and allRedPieces	(Setters for allWhitePieces and allRedPieces are not needed.)

4.2 Package main

4.2.1 class Main

This class is the main program. You don't have to implement anything in this class. You can test the program by running this class.

5. Finished Code Run Example

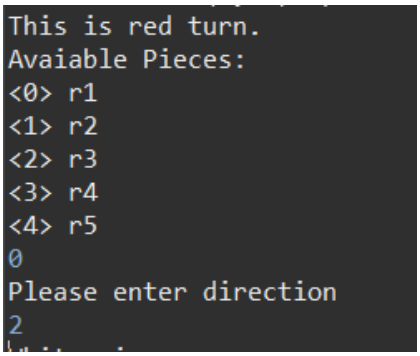
5.1. Start the game

```
Welcome to zerg chess!
White pieces:
BaseUnit w1: (0, 0) hp: 2
BaseUnit w2: (1, 0) hp: 2
BaseUnit w3: (2, 0) hp: 2
BaseUnit w4: (3, 0) hp: 2
BaseUnit w5: (4, 0) hp: 2
Red pieces:
BaseUnit r1: (0, 4) hp: 2
BaseUnit r2: (1, 4) hp: 2
BaseUnit r3: (2, 4) hp: 2
BaseUnit r4: (3, 4) hp: 2
BaseUnit r5: (4, 4) hp: 2
```

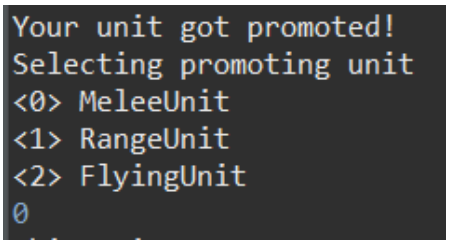
5.2. White turn interface

```
This is white turn.
Avaiable Pieces:
<0> w1
<1> w2
<2> w3
<3> w4
<4> w5
0
Please enter direction
0
```

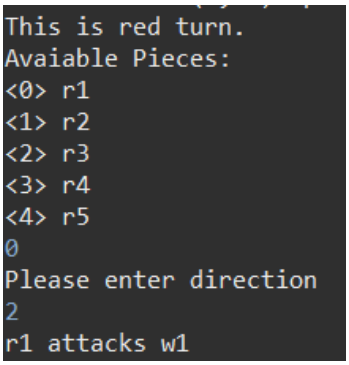
5.3. Red turn interface



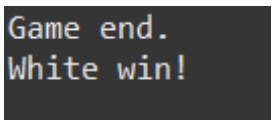
5.4. Promote interface



5.5. Attack interface



5.6. Game ending interface



6. Score Criteria

1 for each test case (total 30 marks)

UML (total 8 marks)

: 1 for each UML class.

: 1 for each correct inheritance.

The total Score will be rounded to 2.5.