**Background**

This game has two players. Each has $20 to buy an army that can be composed of three different kinds of ﬁghter units: soldiers, archers, and cavalry. The following table gives a summary of each unit. The column denoted by Fighter indicates the kind of ﬁghter unit, Life the starting number of life points, Experience the starting number of experience points, Speed the formula used to compute its speed, Damage on attack the formula used to compute the damage it inﬂicts on attack, Lost life after defense the formula used to compute the number of life points it loses after defending itself from the damage inﬂicted by another unit’s attack, and the Cost for buying such a unit.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Fighter | Life | Experience | Speed | Damage on attack | Lost life after defense | cost |
| Soldier | 3 | 0 | 1+experience | 1+experience | If damage>experience: -1 life | 1 |
| Archer | 3 | 0 | 3 | 1+experience | -1 life | 2 |
| Cavalry | 4 | 0 | 2 | 2\*experience+1 | If damage>experience/2: -1 life | 3 |

After each player buys as many units as it wants with the $20 available, the army of each player takes positions to get ready for battle as follows: all soldiers are positioned ﬁrst, then archers, then the cavalry (think about a stack in which all soldiers are at the top, the archers in the middle, and the cavalry at the bottom). Once each army is in position, battle proceeds as follows. The ﬁrst alive unit of each army (i.e., that at the top of the stack) gets into combat. Combat between two units (say U1 and U2) proceeds as follows: • If the speed of one (say U1) is greater than that of U2, U1 attacks ﬁrst inﬂicting some damage (whose value is given by the Damage on attack formula in the table), U2 defends (possibly avoiding losing 1 life, as indicated by the Lost life after defense formula in the table), and if U2 is still alive after this, then U2 attacks, and U1 defends. Note that archers are poor defenders and unconditionally lose a life when they are attacked. • If the speeds of U1 and U2 are identical, then both attack at the same time. The main difference then is that now both have to defend from the damage inﬂicted by the other unit (regardless of whether the other unit died as a consequence of the attack), while in the above case, U1 will not take damage from U2 if U2 dies after the attack.

**Task 1**

1. Create three classes called Soldier, Archer, and Cavalry. Each of these classes have two instance integer variables life and experience, and the following methods:

• isAlive(): it returns true if the ﬁghter’s life is greater than 0, false otherwise. • loseLife(lostLife): it decreases the life of the unit by the amount indicated by lostLife (which is assumed to be positive). • gainExperience(gainedExperience): it increases the experience of the unit by the amount indicated by gainedExperience (which is assumed to be positive). • getCost(): it returns the cost of purchasing this unit. Note that the method is static and, therefore, it cannot use the value of instance variables (only of static variables). • getSpeed(): it returns the speed of the unit. • attack(): it returns the amount of damage performed by the unit when it attacks • defend(damage): it decreases the life of the unit (if required) depending on the value of damage • \_\_str\_\_(): it returns a string indicating the type of unit, its current life and experience.

2. Write a Python program that allows a user to create objects of each of the classes Soldier, Archer, and Cavalry, and test each of the functionalities of the methods of the classes.

**Task 2**

1. Write a class Army which contains the name of the player and a stack. When you create an object of this class you will need to

fit1008 introduction to computer science practical session 8 3

provide the player’s name as an argument. The initialization method will display the following message for a player with associated name Player Name:

Player Name choose your army as S A C where S is the number of soldiers A is the number of archers and C is the number of cavalry

Once this is done, the method will read the input, ensure three positive numbers (i.e. ≥0) are given, and push each purchased object into the player’s stack. In doing this it will make sure that all cavalry (if any) is pushed ﬁrst, then every archer (if any), and ﬁnally every soldier (if any). It will also make sure that the player did not spend more than the alloted treasury amount (i.e., $20). If the input is in any way invalid, the player should be asked to provide it again.

2. Write a Python program that allows two players to set up their armies.

**Task 3**

Extend the program your wrote for Task 2. Add a method gladiatorialCombat(Player1\_army, Player2\_army) to deal with combat in a “gladiatorial” way, i.e., the ﬁrst ﬁghter for each army will ﬁght until it either dies, or kills every ﬁghter in the other army. In order to implement this, your method should follow the following steps. While each army has at least one ﬁghter unit in the stack:

1. pop a ﬁghter from each army (say U1 and U2),

2. attack and defend following the rules indicated in the Background Section

3. If at the end of combat (after attack and defense has happened) both units are still alive, they both lose one life; else the unit who remains alive (if any) gains one experience point.

4. Every unit that is still alive after step 3 is completed gets pushed back into the stack ready to ﬁght in the next combat 5. Go back to step 1

Once at least one army is empty, indicate who the winner is (if any) and print the remaining elements in its army stack. If both stacks are empty, print a message indicating the game is a draw.