FIT9133 Assignment #1
Building a Simple Combat Simulator with Python Programming
Semester 2 2018

Gavin Kroeger
Admin Tutor, Faculty of IT
Email: Gavin.Kroeger@monash.edu

August 2, 2018

Revision Status

```
$Id: FIT9133-Assignment-01.tex, Version 1.0 2018/07/18 20:30 pm Gavin Kroeger $
$Id: FIT9133-Assignment-01.tex, Version 1.1 2018/07/27 18:30 pm Jojo Wong $
$Id: FIT9133-Assignment-01.tex, Version 1.2 2018/07/31 19:00 pm Gavin Kroeger $
$Id: FIT9133-Assignment-01.tex, Version 1.3 2018/08/02 15:25 pm Jojo Wong $
```

Contents

1 Introduction

This assignment is due on **24th August 2018 (Friday) 5pm**. It is worth **15% of the total unit marks**. A penalty of 5% per day will apply for late submission. Refer to the FIT9133 Unit Guide for the policy on extensions or special considerations.

Note that this is **an individual assignment** and **must be your own work**. Please pay attention to Section 4.2 of this document on the university policies for the *Academic Integrity, Plagiarism and Collusion*.

This first assignment consists of **two main tasks** and each task should be submitted as a separate Python program with supporting documentation on its usage. All the program files and any supporting documents should be compressed into one single file for submission. (The submission details are given in Section 4.)

Assessment: Both tasks (Task 1 and Task 2) carry equal weightage of marks with 50% **each.**

2 The Combat Game

Your task is to implement a simple battle simulator that pits one army against another.

At the start of the game, each commander is given a starting total of $10. Units are purchased and stored in their army. The commanders may spend as much or as little of their money as they desire. After the armies are assembled, the units are then made to fight each other **in the order they were purchased in**. Each unit in the standard game costs $1.

There are three types of units available:

- Archer
- Soldier
- Knight

Each unit has a weakness and a strength. Archers are good against Soldiers but are terrible against Knights. Soldiers are good against Knights but can't win against Archers. Knights beat Archers, but fall short against Soldiers. If a unit comes up against a unit of the same type, both lose. Table 1 indicates who wins in any encounter.

Type of Unit	Archer	Soldier	Knight
Archer	Tie	Archer	Knight
Soldier	Archer	Tie	Soldier
Knight	Knight	Soldier	Tie

Table 1: Outcomes of battles when units enter combat.

After each fight, the winner is left on the battlefield to fight the next combatant. If both units lose, then two new units are taken from the army and begin their fight.

2.1 Task 1: The Basic Game

The basic implemenation of the game should address the following requirements:

- Each army should be stored as a Python list in the order that they were purchased. This order must be maintained so that the units fight at the correct time.
- You do not require a graphical interface but will require a menu and allow user interaction through the console (i.e. command-line interface).
- Combat should be resolved automatically. The outcome of each fight is listed in the console until one army is defeated. Once this occurs, the winner is listed and the game ends.

You may make use of the following Python built-in libraries if you wish:

- Math
- String
- Random

You MAY use these libraries. However, their use is not needed to complete the assignment! You may not make use of any other libraries to complete this assignment.

NOTE: You should name this basic implementation as: "**basic_game_StudentID.py**".

2.2 Task 2: The Extended Game

You must implement **TWO** of following upgrades to your basic game:

1. **Improved Combat**

   Combat is very simple in the original game, you can try to implement an improved version of the combat by using the following rules:

   - In the cases where a unit would win, it instead deals its damage before the other unit is able to deal theirs. Soldiers hit Knights first, Archers hit Soldiers first, and Knights hit Archers first.
   - Knights are able to trample other units. If the Knight is fighting an Archer and the unit behind the Archer is another Archer, then the Knight deals its damage to both Archers.
   - If an Archer is at the front of its army but not in battle, they deal their damage to the opposing unit if they are still alive at the end of combat.

   The changes made here effectively change the order of combat which may impact further based on choices below.

2. **Health**

   Units have health rather than just losing by fighting. Knights have 3 health, Soldiers have 2, and Archers have 1. Each unit now deals damage based on who they have an advantage over. If the unit has the advantage, they deal 3 damage, if the unit has the disadvantage, they deal 1, and if the unit has no advantage or disadvantage, they deal 2.

   If a unit is still alive at the end of combat, they remain on the battlefield. You will need to consider how health is dealt with if improved combat is implemented. This could mean units are killed before their damage can be dealt in retaliation.

3. **Medics**

   Money remaining after the purchasing of armies will be used to hire and outfit medics. When a unit dies, it will be returned to the pool at the back of the army. Each time this happens, supplies for the medics decreases. Once the medics have no supplies left, they will be unable to save any more units.

   Medics are hired and supplied at $1 per unit. All money at the end of army creation is spent on Medics.

4. **Expanded Armies**

   Add two new units for the commanders to choose from:

   - **Siege Equipment:** who win against everyone except Knights and Wizards.
   - **Wizard:** who can beat anything, but they can't dodge Archer arrows.

   Depending on your choices for other upgrades, you may wish to give these units different amounts of health.

Depending on which changes you make, you WILL need to re-adjust the prices of units (based on the strength of each unit type).

NOTE: You should name the extended version as: "**extended_game_StudentID.py**".

3 Important Notes

3.1 Documentation

Commenting your code is essential as part of the assessment criteria (refer to Section 3.2). You should also include comments at the beginning of your program file, which specify your name, your Student ID, the start date and the last modified date of the program, as well as with a high-level description of the program. In-line comments within the program are also part of the required documentation.

3.2 Marking Criteria

The assessment of this assignment will be based on the following marking criteria. The same marking criteria will be applied on both tasks:

- 60% for working program — functionality (for both tasks);
- 10% for code architecture — algorithms, data types, control structures, and use of libraries;
- 10% for coding style — clear logic, clarity in variable names, and readability;
- 20% for documentation — program comments and user documentation.

4 Submission

There will be NO hard copy submission required for this assignment. You are required to submit your assignment as a **.zip** file name with your Student ID. For example, if your Student ID is **12345678**, you would submit a zipped file named "**A1_12345678.zip**". Note that marks will be deducted if this requirement is not strictly complied with.

Your submission must be done via the assignment submission link on the FIT9133 S2 2018 Moodle site by the deadline specified in Section 1, i.e. **24th August 2018 (Friday) 5pm**.

4.1 Deliverables

Your submission should contain the following documents:

- At least the two Python scripts named as "**basic_game_StudentID.py**" and "**extended_game_StudentID.py**". If you have more than two scripts for your implementation, name them with the same convention.
- A report (in the form of a PDF) that demonstrates how to use your program through the use of text and screenshots. You must detail which choices you made to extend your program.

  NOTE: Your programs must at least run on the computers in the University's computer labs. Any submission that does not run accordingly will receive no marks.
- Electronic copies of ALL your files that are needed to run your programs.

Marks will deducted for any of these requirements that are not strictly complied with.

4.2 Academic Integrity: Plagiarism and Collusion

Plagiarism Plagiarism means to take and use another person's ideas and or manner of expressing them and to pass them off as your own by failing to give appropriate acknowledgement. This includes materials sourced from the Internet, staff, other students, and from published and unpublished works.

Collusion Collusion means unauthorised collaboration on assessable work (written, oral, or practical) with other people. This occurs when you present group work as your own or as the work of another person. Collusion may be with another Monash student or with people

or students external to the University. This applies to work assessed by Monash or another university.

It is your responsibility to make yourself familiar with the University's policies and procedures in the event of suspected breaches of academic integrity. (Note: Students will be asked to attend an interview should such a situation is detected.)

The University's policies are available at: `http://www.monash.edu/students/academic/policies/academic-integrity`