**TABLE OF CONTENTS**

1. Introduction 2
2. Task 1: The Basic Game 3
3. Task 2: The Extended Game 7
4. Conclusion 12

**INTRODUCTION**

The assignment zip includes three files –

1. basic\_game\_29525756.py
2. extended\_game\_29525756.py
3. FIT9133 Assignment 1 Report-29525756.pdf

The basic\_game\_29525756.py file is the assignment solution to the Assignment Task 1: The Basic Game. The extended\_game\_29525756.py file is the assignment solution to the Assignment Task 2: The Extended Game. This report is a detailed summarization to the aforementioned two assignment tasks.

The details about the solutions to the two tasks are mentioned in their respective sections in the report. Reasons for choosing the respective path and algorithm for the particular task are also mentioned. Screenshots and directions on how to go through the game are provided along with it.

Python version used: Python 3.6.5.

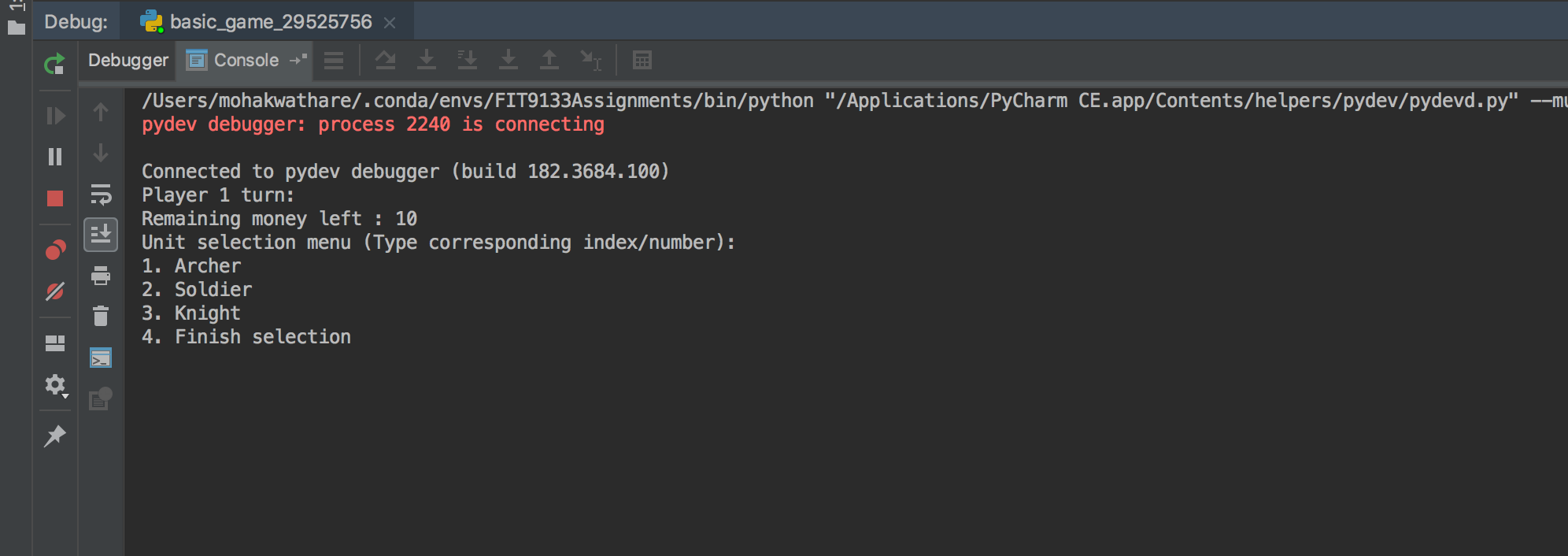
**TASK 1: THE BASIC GAME**

**2.1 DECLARATIONS**

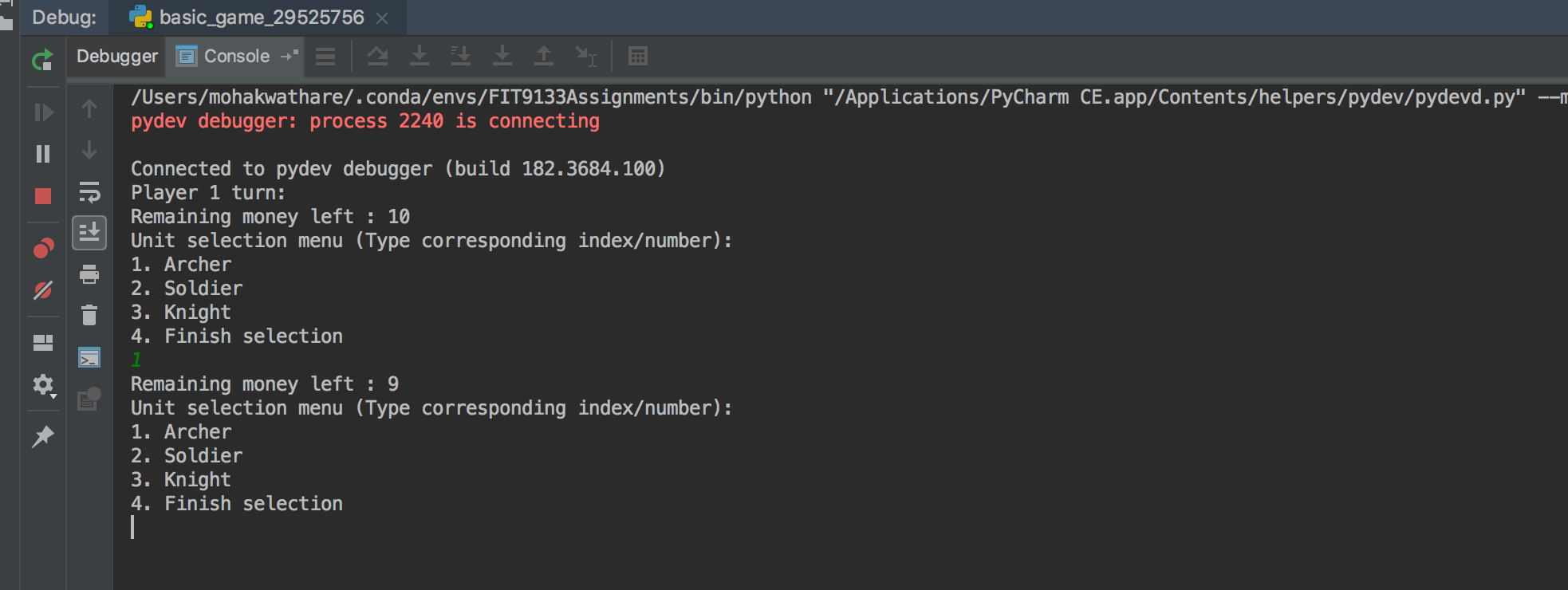
This task has been implemented in the basic\_game\_29525756.py file present in the zip. This python file includes the basic aspect of the combat game implemented according to the given task.

**2.2 WALKTHROUGH**

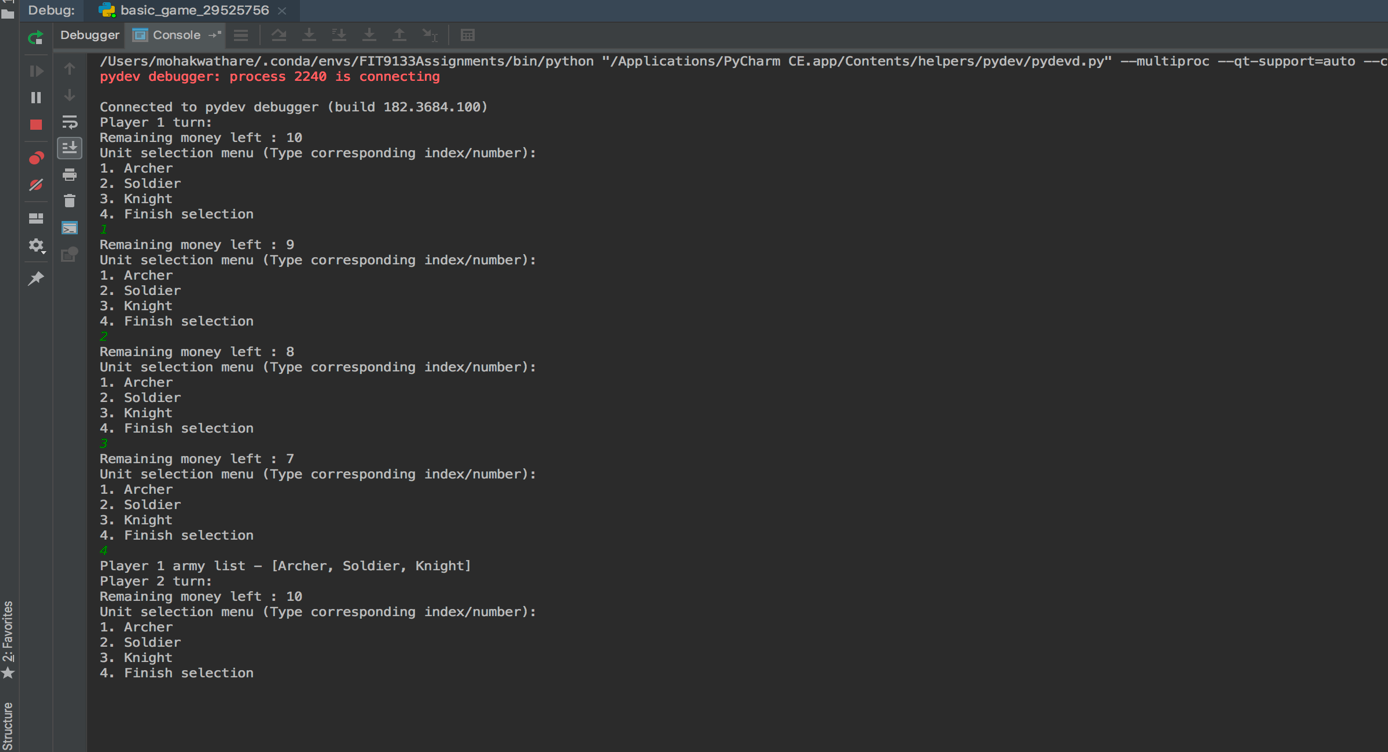
The implementation starts with the selection of the armies according to the choice of the player. The player is shown their remaining money at each stage of the selection process. The player is prompted to select his army from the options given to them (1 for selecting Archer, 2 for selecting Soldier, 3 for selecting Knight or 4 for finishing the selection process).

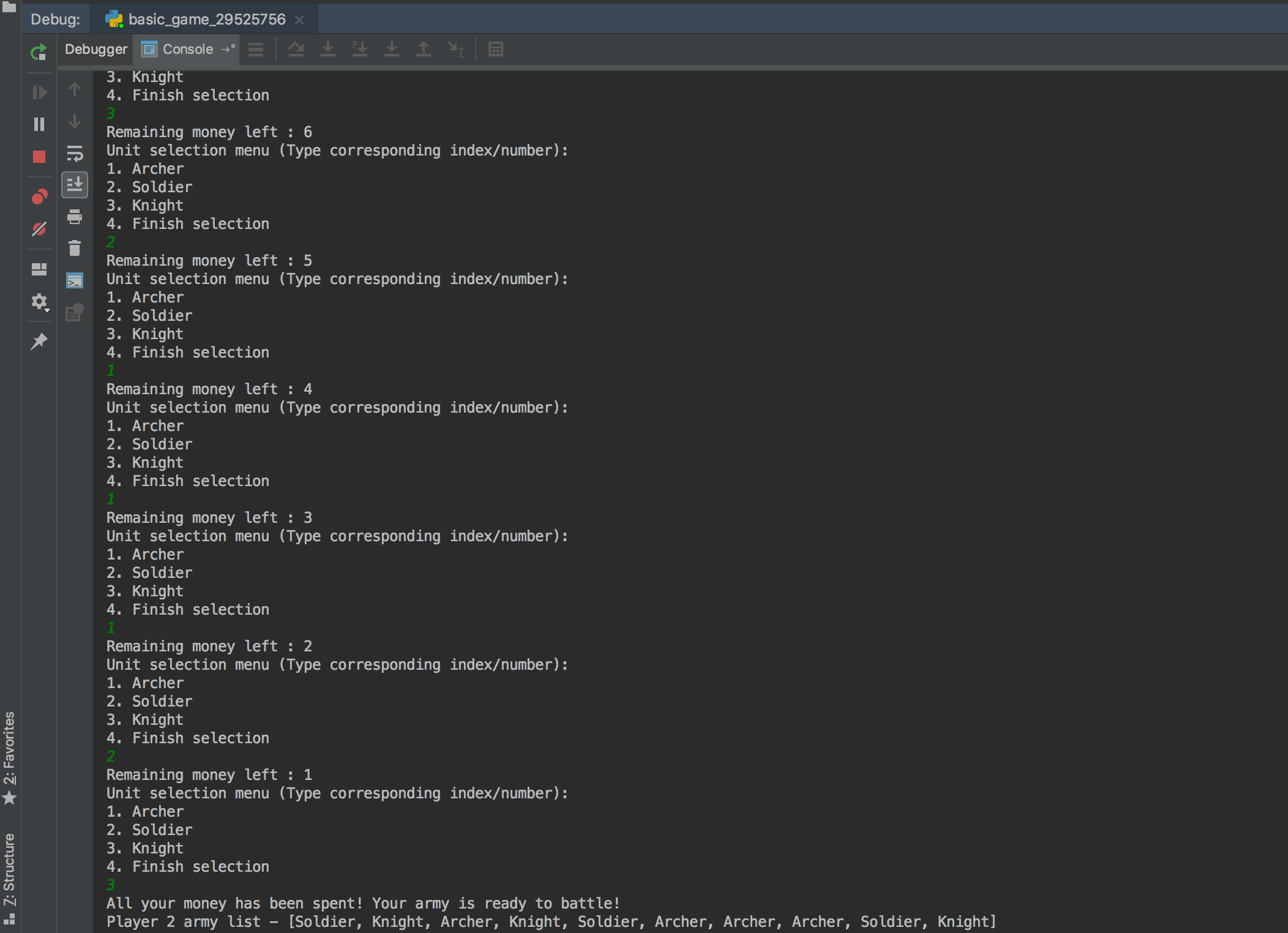


After this prompt the player selects his unit, which is then placed at the start of a list comprising of the players’ units. Post selection of unit, the money remaining with the player is reduced by 1 – the cost of each unit.

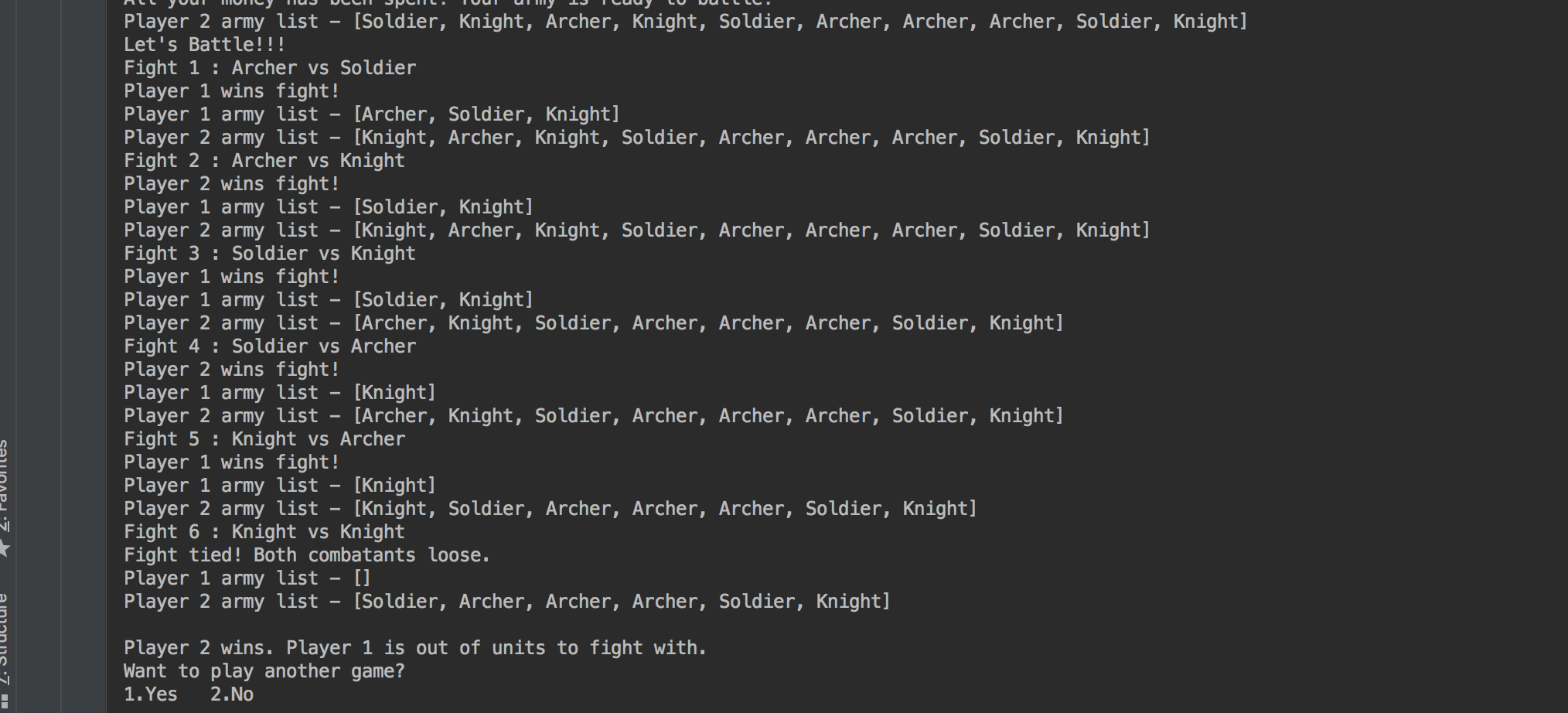


The unit selection process continues till the player has exhausted their resources/money or they wish to exit the selection process by entering ‘4’ which finished the selection process as specified in the selection menu. At the end of the selection the selected army is displayed and the other player can select their army.





After this the program takes over and simulates the battle according to the advantage – disadvantage chart given in the instructions for the game rules. The battle takes place with each sequential fight between the two units from each team. If a unit wins, it stays to battle the next unit in line from the other team. Example – in Fight 1, Archer wins against Soldier of Team 2. Hence, it stays in the battlefield to battle in the next in line from the player 2 army i.e. Knight. And that is the Fight 2. The game progresses further in this manner.

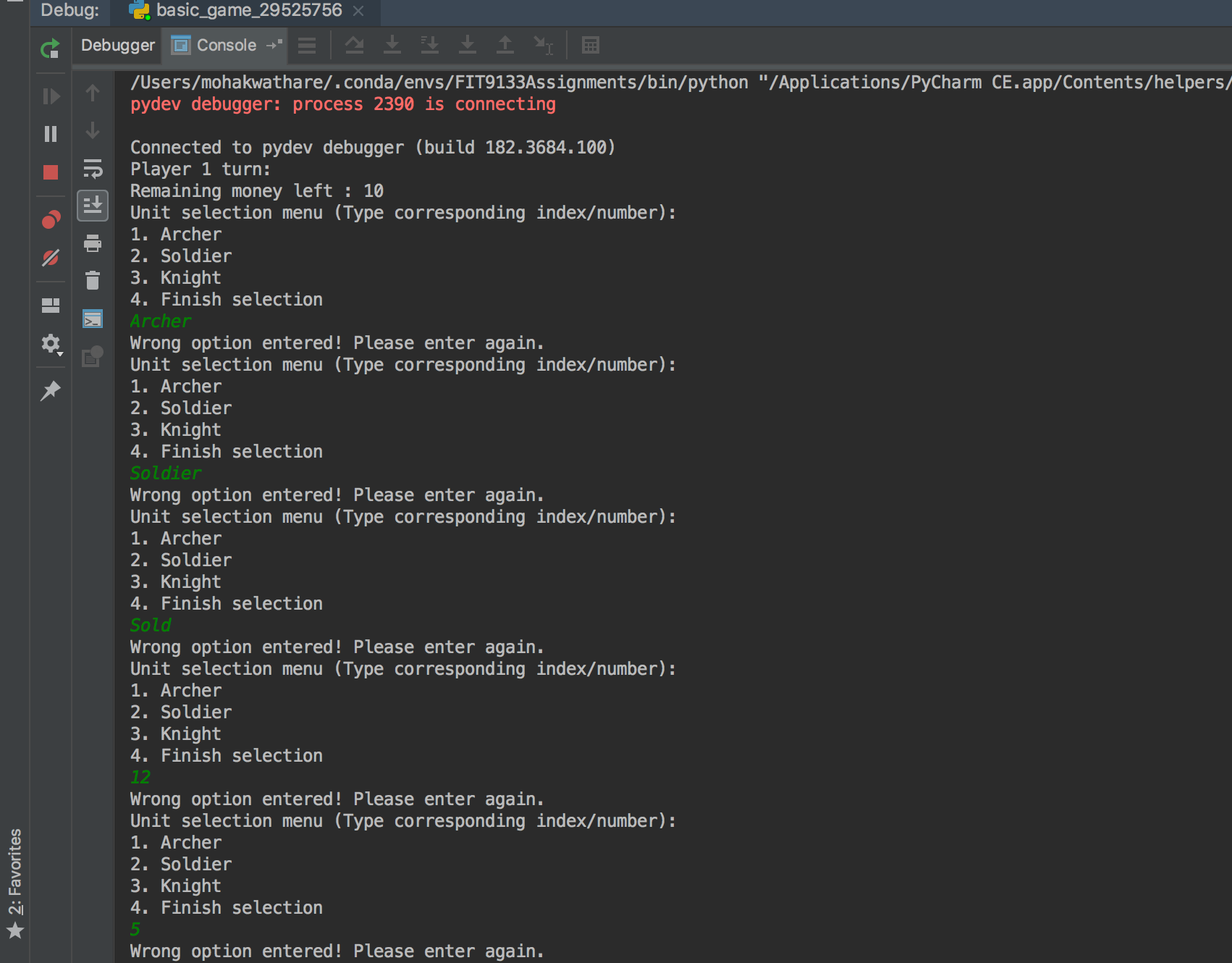


The army list is shown at the end of each fight to show how the players stand after each fight, showing their team lists. After all the simulations are done, the team with no remaining soldiers left to fight loses and the other team is declared the winner.

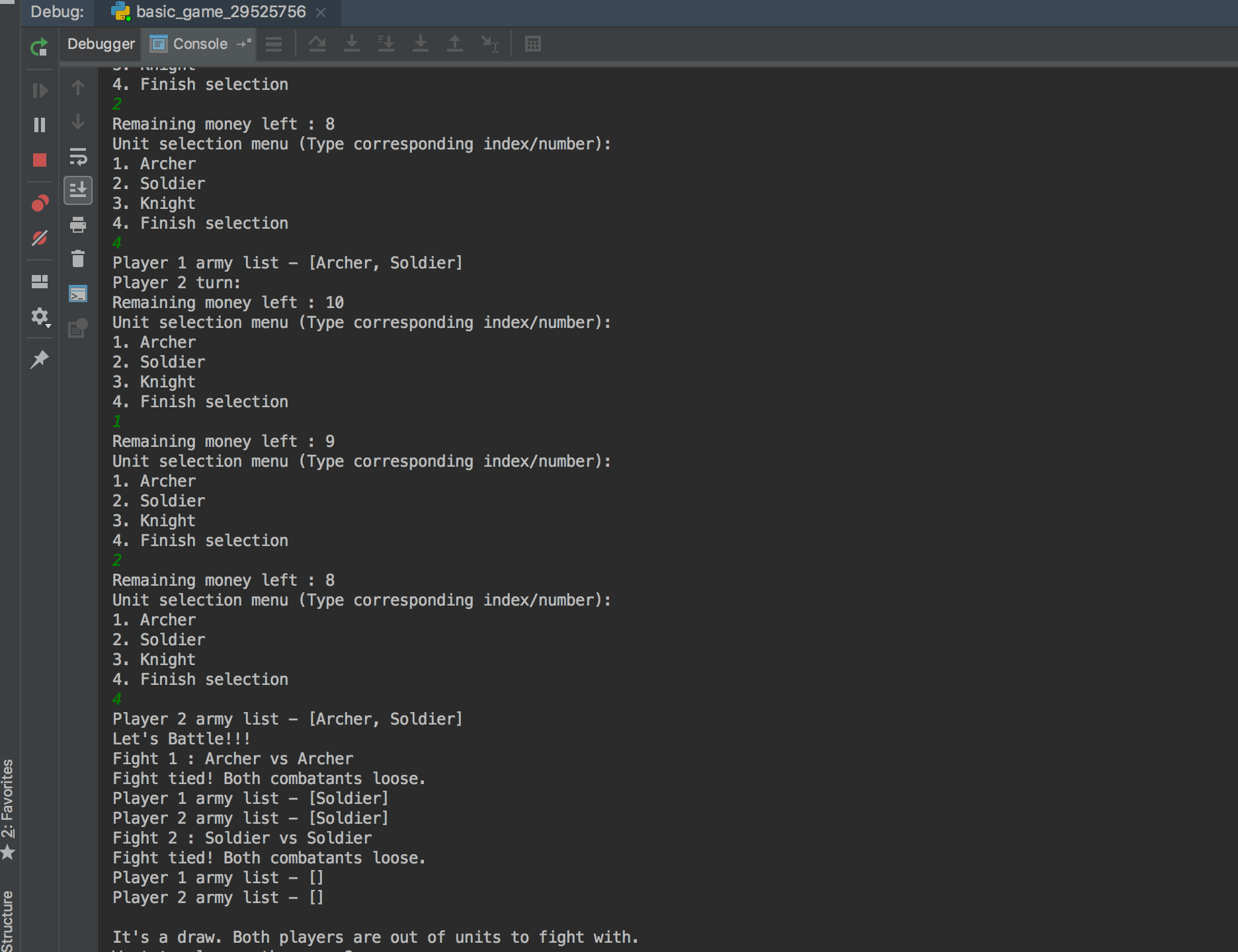
The user can go ahead and simulate another game if they wish by entering ‘1’ when prompted with the “Want to play another game?” question. Or they can lay it all to rest by entering ‘2’.

**2.3 FAIL SAFES**

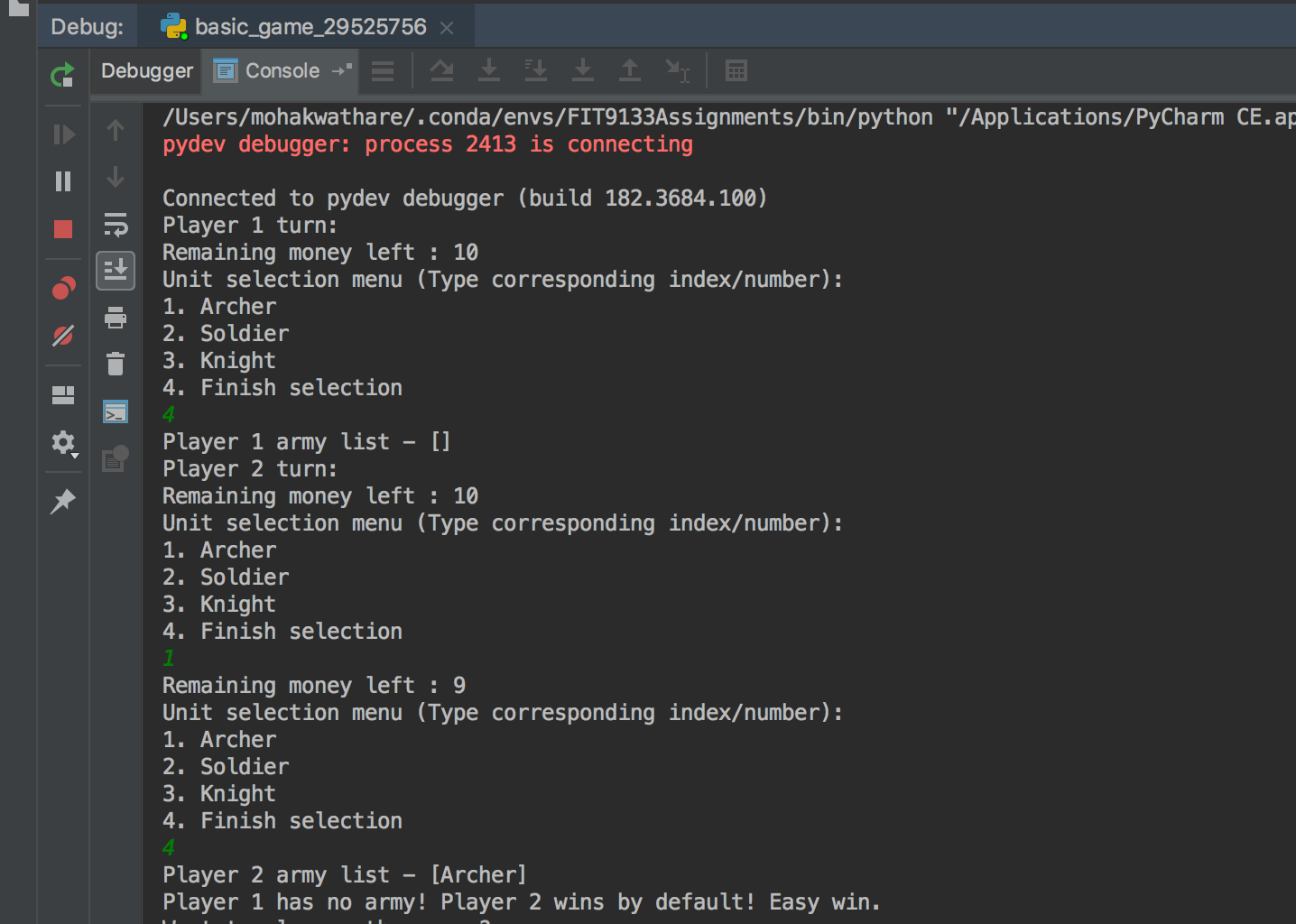
The program will not take any other inputs for unit selections other than the already specified 1-4. It’ll prompt the player to enter again until it enters the correct option or exits by entering ‘4’ to quit unit selection menu.



In the case of a complete draw battle i.e. when both the players run out of units to fight with, the battle is declared as a draw.



In the case either player do not buy any units (want peace) for their team then the battle is ruled in the favour of the player who has bought some units. If both players do not buy any units, then the match is declared as a draw as both of the players supposedly want peace.



**TASK 2: THE EXTENDED GAME**

**3.1 DECLARATIONS**

This task has been implemented in the extended\_game\_29525756.py file present in the zip. This python file includes the extended game aspect of the combat game implemented according to the given task.

The two upgrades implemented in this task are –

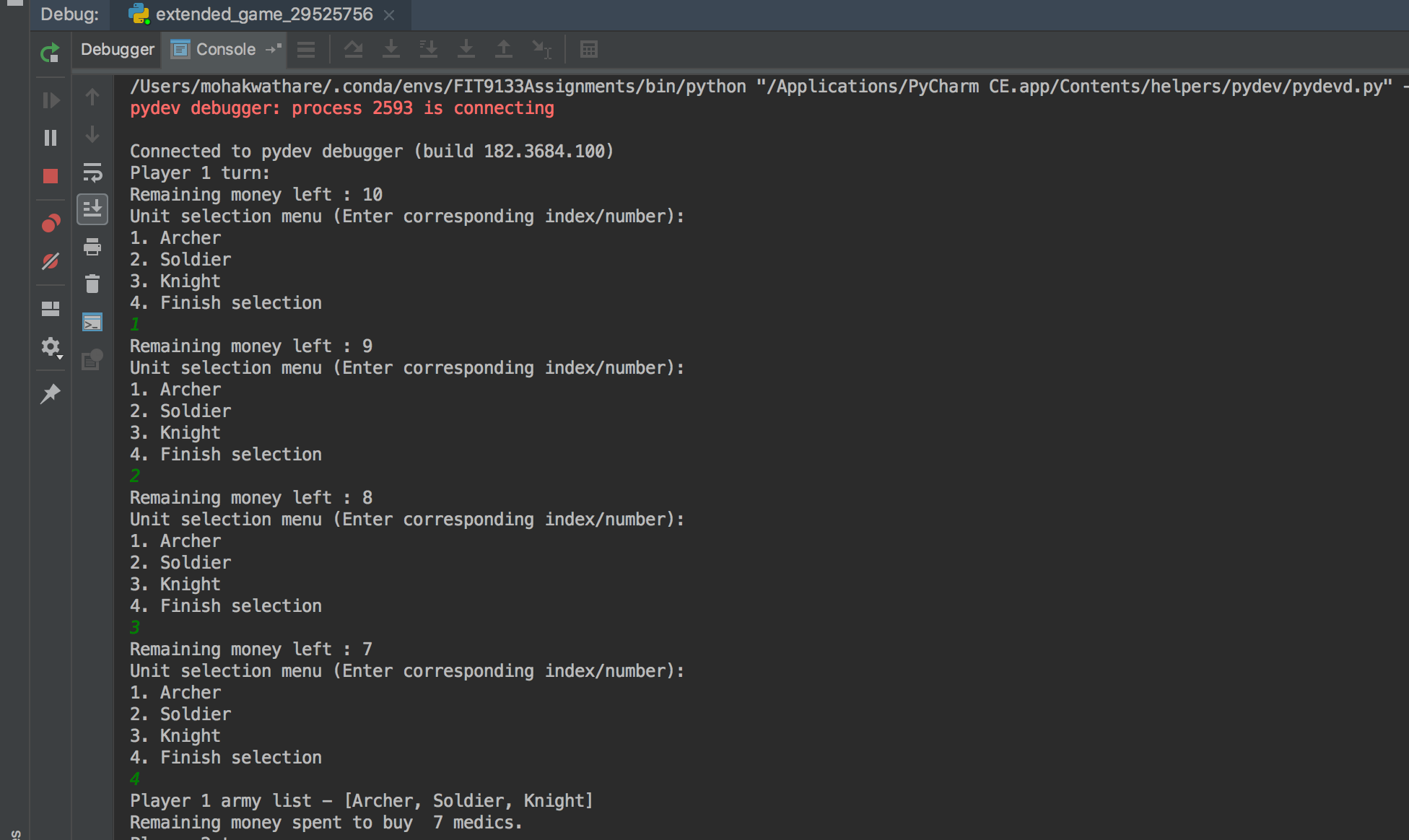
1. Health
2. Medics

The two particular upgrades were chosen as they act as an entire package towards the healing aspect of the game. They go hand in hand towards the notion of healing/reviving an individual unit. The health upgrade adds additional health and damage points to the units while the medics upgrade gives the chance to revive units and adds them at the back of the battle line. They are the most matching and complete a single “heal” package when put together.

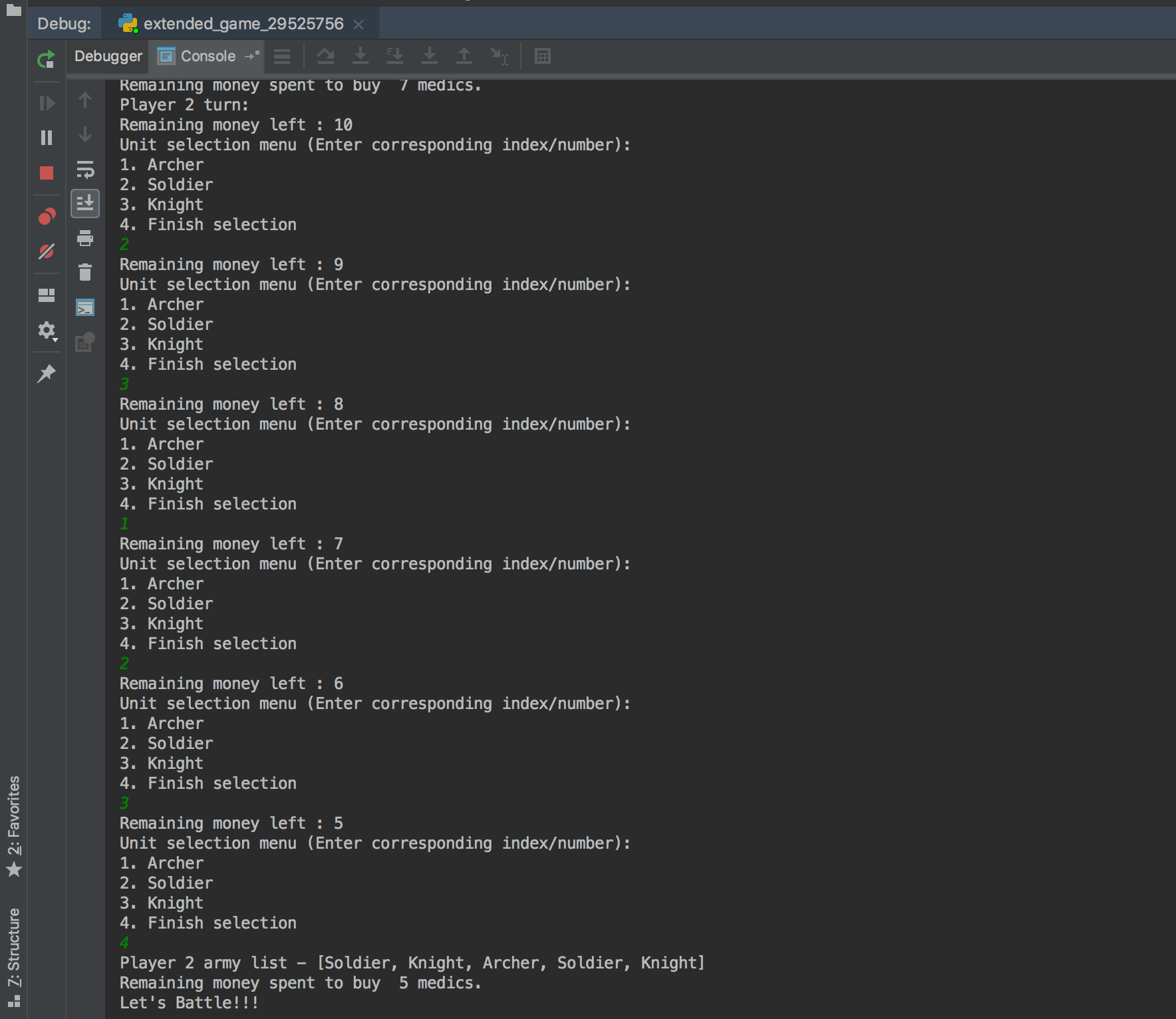
The cost of the units has not changed from the basic 1$ per unit as all the units still have the same advantage/disadvantage over other units and as result stand at the same spot. Furthermore, archers seeming to be a low powered unit can deal more damage to units it has an advantage against (knight) and kill it completely, hence putting it on the same power level as other units. Same goes for other units. Hence, all the units have the same price.

**3.2 WALKTHROUGH**

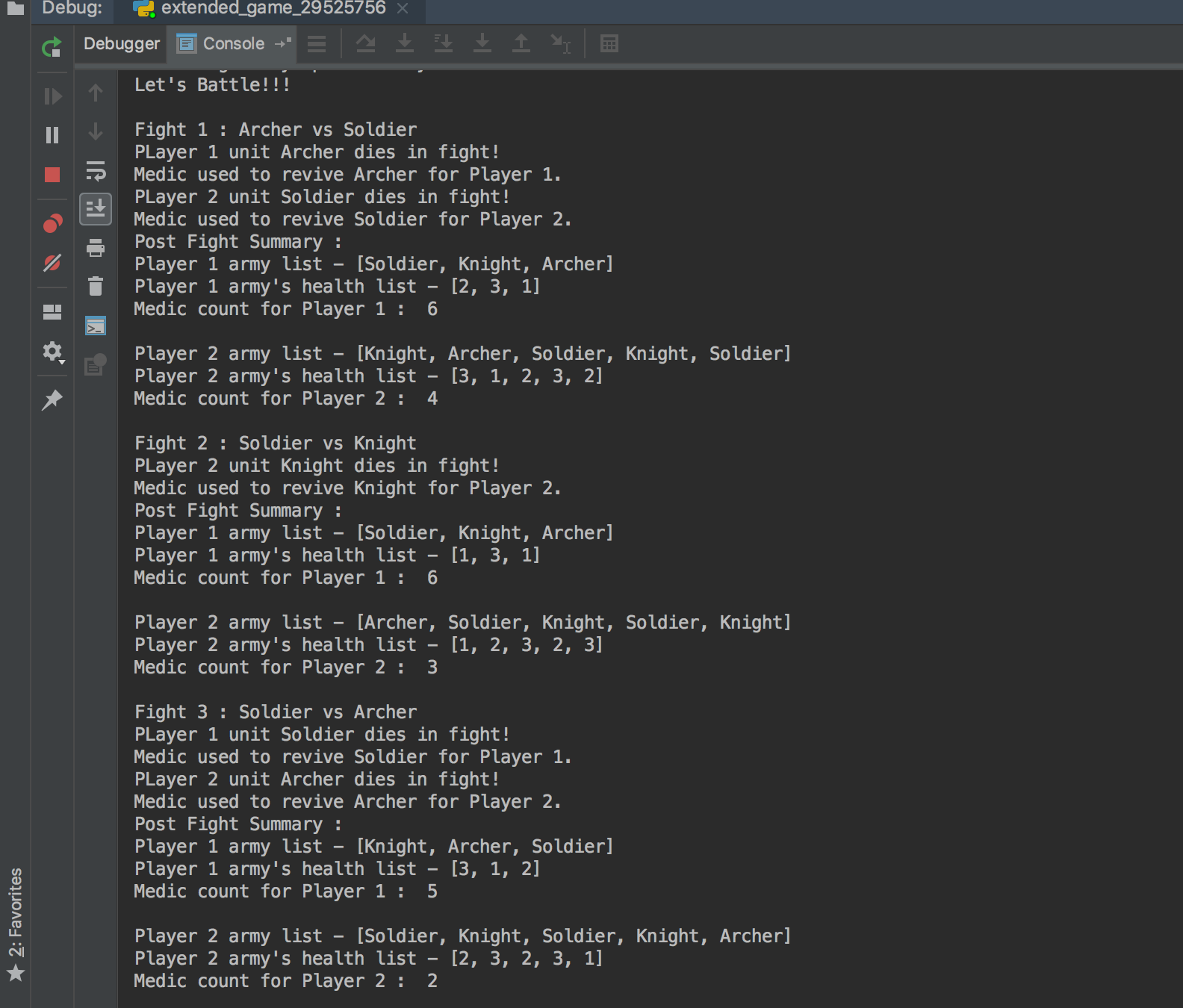
The implementation starts with the selection of the armies according to the choice of the player. The player is shown their remaining money at each stage of the selection process. The player is prompted to select his army from the options given to them (1 for selecting Archer, 2 for selecting Soldier, 3 for selecting Knight or 4 for finishing the selection process). It’s the same menu as the one in the basic game.



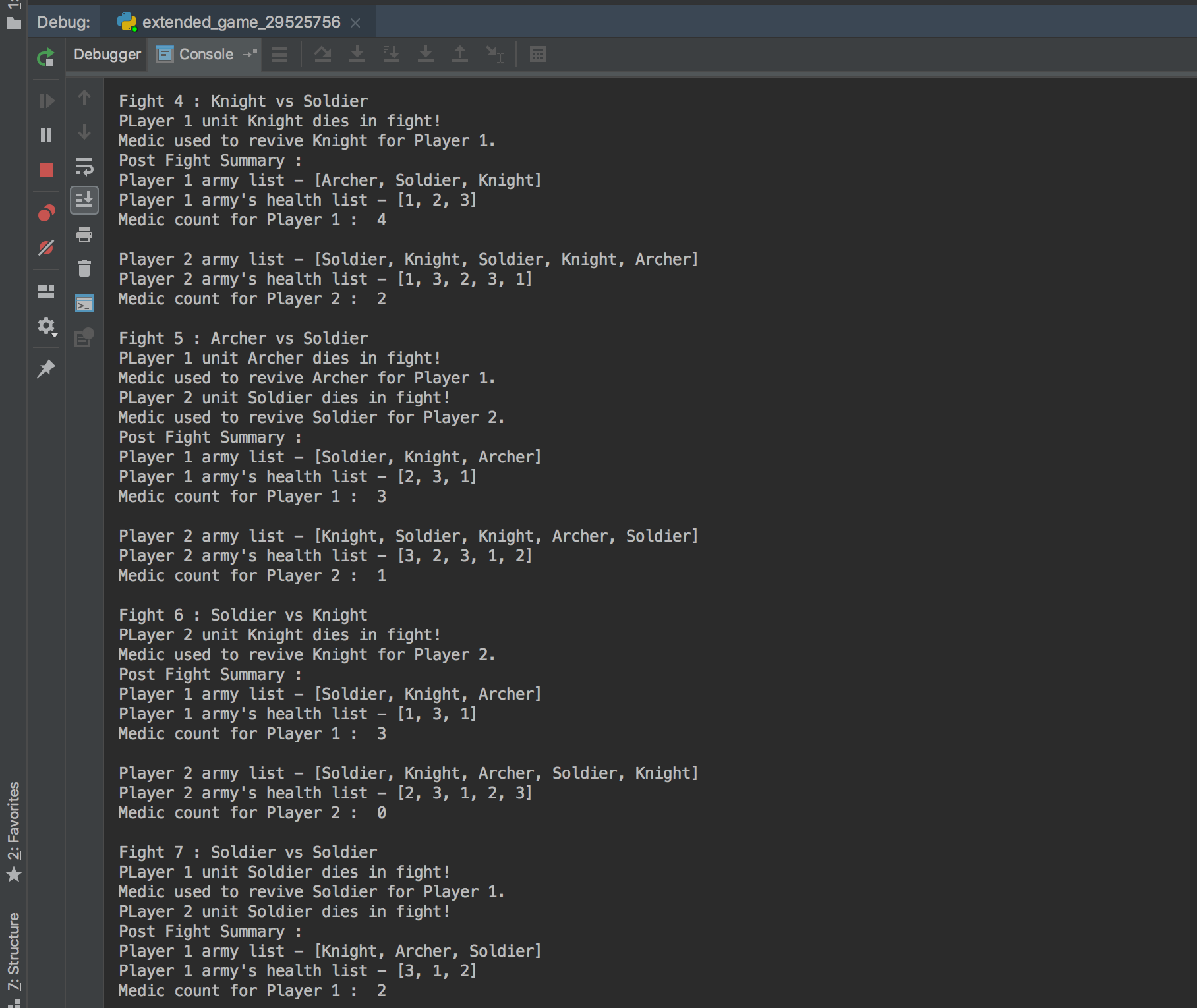
After the player is done with selecting the units for their team, the remaining money is spent to buy medics. In this example, the remaining money left is 7$ after purchase of three units, which bought the player 7 medics. After the other player is done buying his units and is allocated his medics depending on the remaining money they are left with.

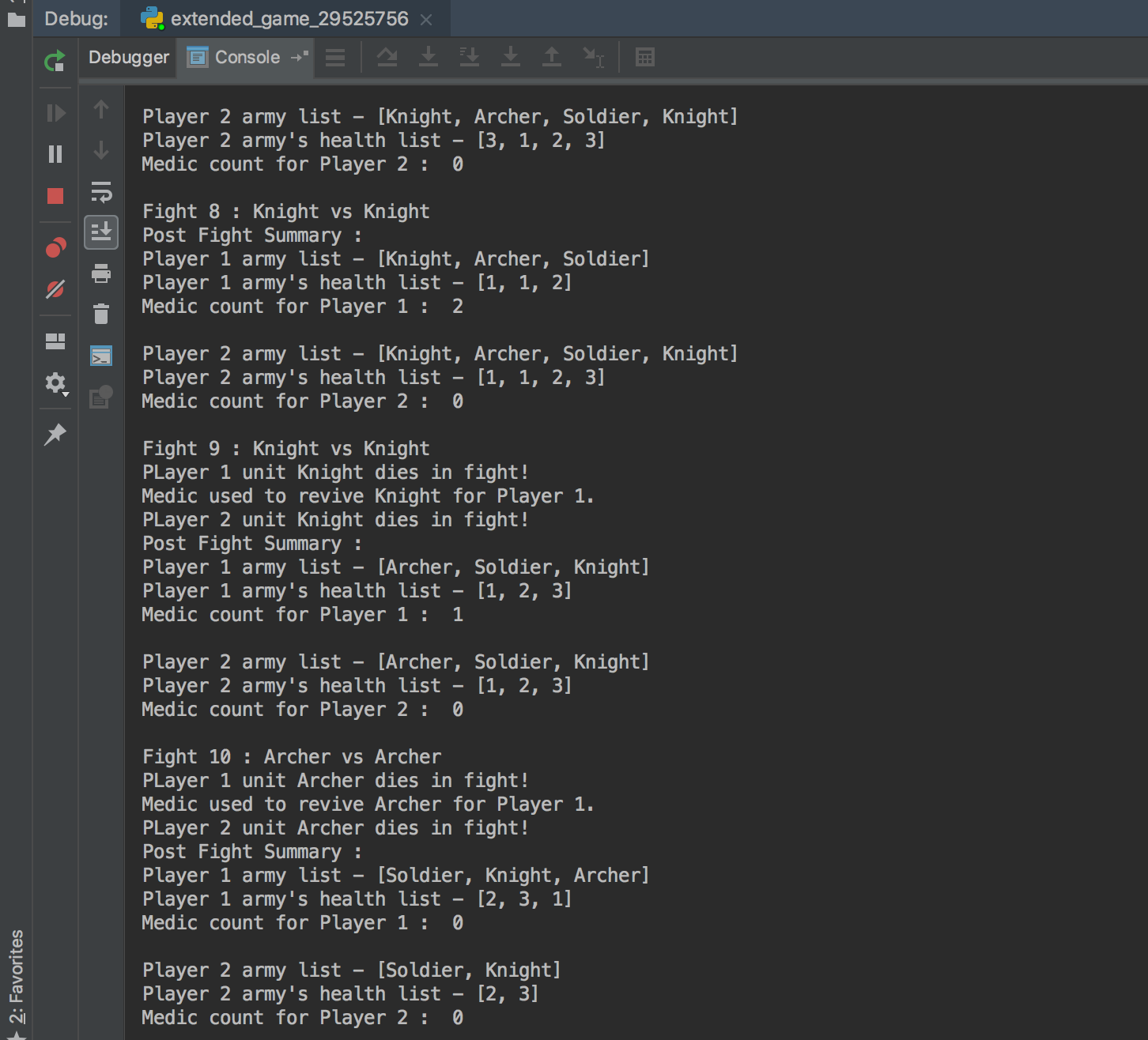


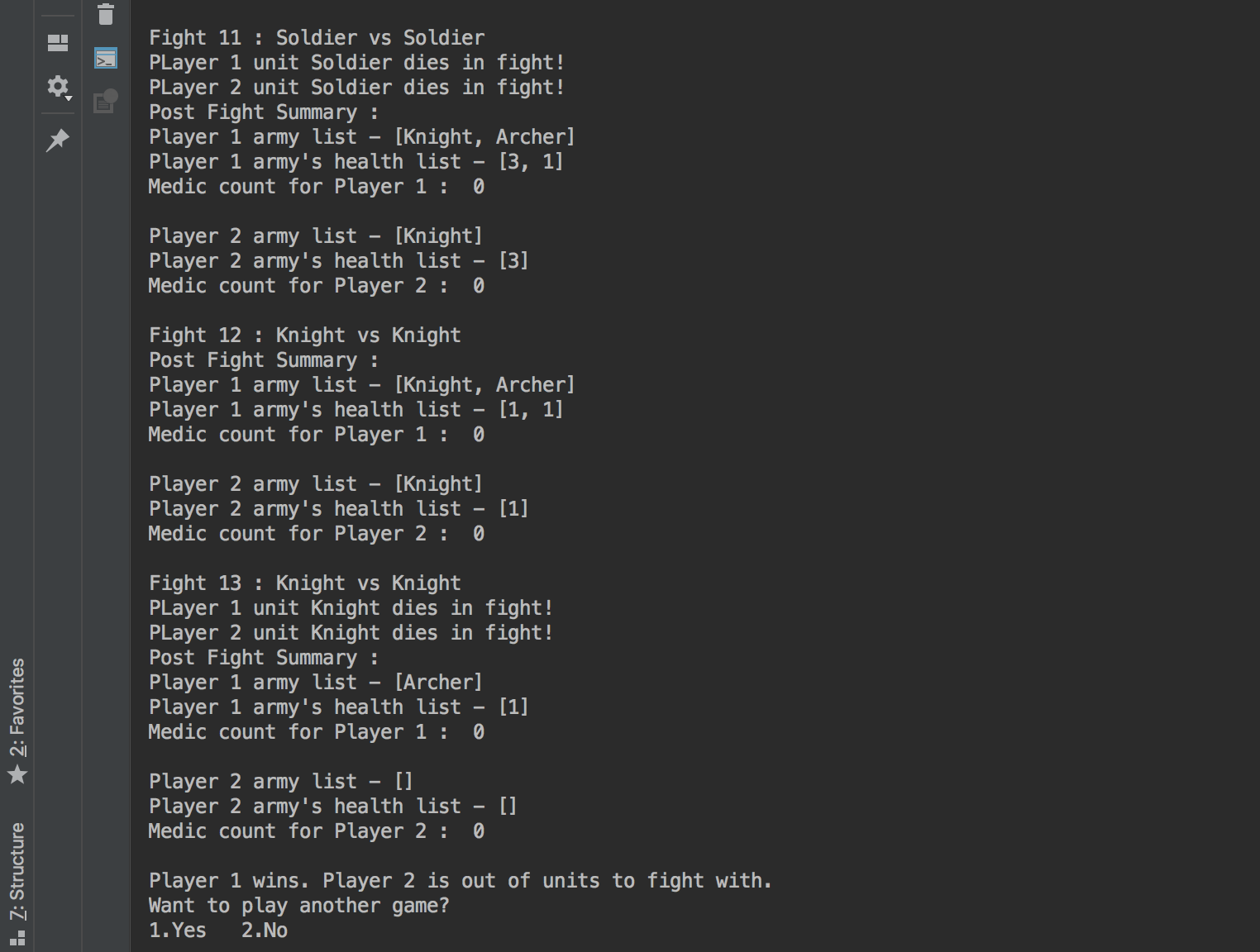
After the unit selection is done, the program simulates the battle. The fight takes place sequentially as it did in the basic game. In this version, the units have a health factor to them (Archer having 1 health, Soldier having 2 and Knight having 3). The unit with the advantage does 3 damage while receiving 1 damage and non-advantage units do 2 damage to each other. Whichever unit has positive health stays in the field. For example – in fight 1 archer is at an advantage over the soldier and hence deals 3 damage and the archer is dealt 1 damage. In the end of the fight the archer whose health is 1 dies as a result. The soldier dies as well at the end of the fight (by being dealt 3 damage his health becomes -1). The post-fight summary is shown with the list of the armies and the list of the health of units in the order of the units in the list. Since both players have medics available, medics are used to revive the dead archer and the dead soldier. Both are placed at the end of the respective armies.



The battle is continued in this manner until one of the armies is over. After this point, whichever army has units remaining wins.







Eventually the battle is over as player 2 is out of units to fight with and medics to revive them. As a result, player 1 wins the battle.

**2.3 FAIL SAFES**

The same fail safes are put in the extended game as the fail safes implemented for the basic game.

**CONCLUSION**

The assignment tasks performed were –

1. The basic game
2. The extended game

The upgrades added to the extended game were –

1. Health
2. Medics

While working on the assignment, the concepts of lists, loops, matrix, data types, functions, string manipulation, if-else statements, etc. were put into practice. These concepts were made use of in a self-learning and application-based way which in turn inculcated deeper understanding of the concepts and how to make use of them in regular python programming tasks.