CSCI3180 – Principles of Programming Languages – Spring 2017

Assignment 3 — Dynamic Scoping and Perl

Deadline: Apr 9, 2017 (Sunday) 23:59

1 Introduction

The purpose of this assignment is to offer you a first experience with Perl, which supports both dynamic scoping and static scoping. Our main focus is dynamic scoping.

The assignment consists of two parts. First, you need to implement a popular card game called "Blackjack" in Perl. Second, you need to implement another game called "Defend Your Kingdom" in Perl with dynamic scoping. For both tasks, the detailed OO designs are given. In the process, you will learn both the programming flexibility and bad readability of codes written with dynamic scoping. All your implementation in Perl should be able to run with Perl v5.18.2, Perl 5, version 18, subversion 2.

2 Task 1: Let's Play Blackjack

In this task, you need to implement the Blackjack game. You should *strictly follow* the specified OO design and game rules for this task. For the OO design, you have to follow the prototypes of the given classes exactly, but can choose to add new member variables and functions.

2.1 Description

There are 52 cards in a deck. Each time we start the game with a new deck of 52 cards. Face cards, including kings, queens and jacks are all counted as 10 points each. Ace can be counted as 1 or 11 points. If there is at least one ace in the player's hand, and this player wants to count at least one ace as 11, then it is a *soft hand*; otherwise it is a *hard hand*. For example, there could be more than one ace in the player's hand, it is a hard hand only when the player counts all aces as 1 points; otherwise it is a soft hand. All other cards are counted using the numeric value shown on the cards. During the whole game, all cards in both players' and dealer's hands are visible to each other. In the Blackjack game, there should only be one dealer and at least one player.

Let us assume there are N players, from player1 to playerN in the game. When the game starts, the dealer will first deal two cards to the players one by one from player1 to playerN. And then the dealer will deal two cards to herself or himself. After all players and the dealer receive their first two cards, then we have the players' sessions for Player1 to PlayerN.

In each player's session, the dealer will ask this player to take one of the following two actions.

1. Hit: Take another card from the dealer;
2. Stand: Take no more cards. If this player chooses to stand, her/his session has been finished. And the game proceeds to the next player.

Each player may hit as many times as desired as long as the total is not above hard 20. On reaching 21 (including soft 21), the player is required to stand automatically without taking any further actions. If the player has the total above hard 21, then this player busts, and he/she can't take any more actions and is required to stand automatically. After a player busts or stands, the game proceeds to the next player.

When the last player finishes her/his session, the dealer stands or hits according to the rule of the game for dealer. After the dealer finishes her/his session, the game reaches the end.

At the end of the game, the dealer will count the total value for all players and herself/himself. When counting the total value for each player, who does not bust, the dealer will use the maximum possible value as long as the value is not more than 21. For example, the cards in one player's hand are (A, A, 5, 3), then the maximum possible value will be 20 with one ace counting as 1 point and the other counting as 11 points.

A player wins in either of the following two scenarios.

1. The player does not bust while the dealer busts;

2. Both the player and the dealer do not bust, and the player has higher total points than those of the dealer.

Otherwise, the dealer wins.

There are many rules variations for the dealer. In this game, we adopt the one, which requires the dealer to keep "hitting" until the cards in hand total 17 points or more (hard hand).

Please note that, in Blackjack, each player only competes with the dealer. So there is no competition among players.

2.2 Participants

One dealer and at least one player.

2.3 Perl Classes

Please follow the classes `Deck`, `Participant`, `Dealer`, `Player`, `HumanPlayer` and `AIPlayer` defined below in your Perl implementation. You are free to add other variables, methods or classes. We would start the program by running: `Blackjack.pl`

1. **Class** `Deck`

   An abstraction of a deck of cards. You have to implement it with the following components:

   - **Instance Variable(s)**

     `cards`

     - This variable is the reference to a one-dimensional array recording remaining cards in the deck. Each card is represented using the character in the face of this card. Suit information can be ignored since it will not affect the value of a card.

   - **Instance Method(s)**

     `new`

     - Instantiate a `Deck` object with a deck of 52 cards without shuffling, and return this object.

     `shuffle`

     - Shuffle all cards in the deck. Please *strictly follow* our design to call this function.

     `fetchOneCardFromTop`

     - Pop and return the card on the top of the deck.

     `getANewDeckOf52Cards`

     - return an array representing a deck with 52 cards.

2. **Class** `Participant`

   A super class representing a participant in the game. You have to implement it with the following components:

   - **Instance Variable(s)**

     `name`

     - This is a variable recording the name of the participant.

     `cards`

     - This variable is the reference to a one-dimensional array recording cards in the participant's hand.

   - **Instance Method(s)**

     `new(name)`

     - Instantiate a `Participant` object with its name, and return this object.

`hit(card)`

- Deal one card to the participant.

`stand`

- Output information showing the participant has chosen to stand.

`displayHand`

- print all cards in the participant's hand.

`getHandValue`

- return an array with two elements: (hardValue, numOfAce), where hardValue is the total value of cards in hand with ace counted as 1 (hard hand), and numOfAce is the number of ace(s) in hand.

`dropAllCards`

- remove all cards in the participant's hand.

3. **Class** `Dealer`

   This class extends super class `Participant`. You have to implement it with the following components:

   - **Instance Variable(s)**

     `deck`

     - This is a `Deck` object.

     `players`

     - This variable is the reference to a one-dimensional array recording all players in the game. This array naturally encodes the order of all players. For example, the first player in the array should act as the "Player1" in our description.

   - **Instance Method(s)**

     `new(name, players)`

     - Instantiate a `Participant` object by initializing its variables in the following way. First initialize the variables name and players with the parameters passed in. Second initialize the variable deck with a Deck object, and shuffle the deck once.

     `start`

     - Start a new game. First, each participant will be dealt two cards. And then the dealer will ask players one by one to finish their sessions. After all players finish their sessions, the dealer will start her/his own session. After the dealer finishes her/his session, the game reaches the end. The dealer will judge who wins the game between the dealer and each player. You need to output very important information in this function, please refer to the output specifications for more details.

     `reset`

     - Reset the game before starting a new game. This function does two actions. First, all participants drop all cards in their hands if any. Second, the dealer needs to change to a new deck of 52 cards, and shuffle the deck once.

4. **Class** `Player`

   This class extends super class `Participant`. You have to implement it with the following components:

   - **Instance Variable(s)**

     `dealer`

     - This is a variable recording the dealer in the game.

   - **Instance Method(s)**

     `setDealer(dealer)`

     - Set the variable dealer.

5. **Class** `HumanPlayer`

   This class extends class `Player`. You have to implement it with the following components:

   - **Instance Method(s)**
     `hitOrStand`
     - Return "hit" or "stand" if the human player chooses to hit or stand respectively.

     In this function, you need to accept input from command line. A valid input is either 'h' or 's', where 'h' means hit and 's' means stand. If a invalid input is detected, you need to require the user to input again until you get a valid one.

6. **Class** `AIPlayer`

   This class extends **class** `Player`. You have to implement it with the following components:

   - **Instance Method(s)**
     `hitOrStand`
     - Return "hit" or "stand" to indicate if the AI player chooses to hit or stand respectively. You can design and implement any smart strategy you want.

2.4 Execution Context

We provide you with the main program, called "Blackjack.pl". Your implementation should be able to run with this main program.

2.5 Use of Random Numbers

Class `Deck` should be the only place where random numbers are needed. We will provide you with "Deck.pm". So you shouldn't call random number generators in your implementation. In the given main program, we set the seed of random number generator to be 10. So by running your program with the same configuration, we should get exactly the same output. Please do not set the seed of random number generator by yourself.

2.6 Input/Output Specification

You need to input and output in the command line.

- **Input**

  There are two places where inputs are needed. First is to configure the game, which is already provided in "Blackjack.pl". Second is in function `hitOrStand` of Class `HumanPlayer` for human to choose whether to hit or stand.

- **Output**

  The only place where we require you to output is in function `start` of Class `Dealer`. At the end of the game, you need to output two things in oder.

  First, you need to output all cards in each player's hand from Player1 to PlayerN, and then output all cards in the dealer's hand. For example, Mary has 4 cards, (2, 5, K, A), in hand at the end of game, and the cards are ordered based on the order of carding dealing. In this case, Mary gets a 2, then a 5, then a K and lastly an A. So you should output "#Mary 2 5 K A" (with items separated by one space) for Mary in a separate line (do not include the quotation marks in your output). You may want to refer to the running example of our demo program.

  Second, you need to output the results of the Blackjack game. In Blackjack, each player only competes with the dealer. So there is a winner between each player and the dealer. You need to output all winners' names for competing in order. For example, there are 2 players, Sandy and Mary, in the game. They are in the same order as when they play the game. And the dealer is named Alice. At the end of the game, if Sandy loses and Mary

wins when competing with the dealer Alice, then you need to output "#Alice Mary" (with items separated by one space) in a separate line (do not include the quotation marks in your output). You may want to refer to the running example of our demo program.

Except from what we require you to output, you can output everything you want everywhere in your program as long as your output does not contain any '#'.

In your program, please avoid input or output anything containing character '#' except where we require you to do so, since '#' is reserved for displaying information used for our marking script. And you need also to avoid making your program sleep, e.g. call sleep function.

Please note that you can not modify "Blackjack.pl" and "Deck.pm". And the implementation of each class needs to be in a separate file with the file name being exactly the same as the class name.

2.7 Grading Criteria

We will design several test cases to test your program. In each test case, we will use a set of inputs to execute your program. After the execution of your program, we will extract those lines starting with '#' and compare them with ours. So please *strictly follow* our design in input, output and calling random number generator.

2.8 An Running Example of Our Demo Program

Figure 1 in the supplementary materials shows a running example of our demo program.

3 Task 2: Defend Your Kingdom

This task is to simulate a battle game using Perl. You need to use dynamic scoping somewhere appropriate and to your advantage in your implementation. With dynamic scoping, you can implicitly affect the behavior of functions in function calling sequences without passing extra parameters. You should *strictly follow* our OO design in you implementation. For the OO design, you have to follow the prototypes of the given classes exactly, but can choose to add new member variables and functions.

3.1 Background

Once upon a time, you, Arathorn, were the King of the richest realm in this world, but the world was not in peace at that time. There was a notorious man named Lapras. People were scared by just hearing his name. Finally, Lapras came to your land with a well-equipped army. He claimed to take over your kingdom, kill your people and depredate all resources. To protect your realm and your people, you decided to have a duel with this evil person. If you win the game, he would leave your kingdom with his army and never come back again.

3.2 Task Description

You are required to simulate the battle between Arathorn and Lapras.

3.3 Description of the Game

At the beginning of the game, players have the same initial settings with 2000 hp (health points) and 500 dp (defense points).

The health points are used to simulate the life of the player. The player is alive only if its hp is larger than 0; otherwise the player is dead. The player can attack its enemy or opponent only when it is alive. The defense points are used to simulate the protection from its equipment against the attack from its enemy. Each attack has certain power, but the damage to the opponent is decided based on both the power of the attack and the defense points of the opponent.

We give a small example for you to understand the process.

Two players, player1 and player2, in the game are initialized with 2000 hp and 500 dp at the very beginning of the battle. There could be many rounds in a battle. And the battle ends when one of the two players is dead. In each round, player1 and player2 attack each other in turn using either sword or black magic. Using black magic has extra effect, which may invoke another sword attack. We assume player1 always attacks first.

Now the battle starts.

In round one, player1 chooses to attack player2 with sword, which has a power of 800 hp. Since the player2 has 500 dp, so the damage to player2 is 800 - 500 = 300 hp. After the attack, player2 will have 1700 hp remained. Since player2 is alive after the attack, so the battle will continue with player2 attacking player1; otherwise the battle ends.

Player2 chooses to attack player1 using black magic. The power of black magic is 600 hp, so the damage should be 600 - 500 = 100 hp. A black magic induces an extra attack with 40% chance, which is another sword attack with power 1000 hp. Let us assume in this round player2 is very lucky to successfully invoke the extra sword attack with power 1000 hp. For the extra attack, the damage should be 1000 - 500 = 500 hp. So in total, the attack from player2 to player1 causes 600 hp damage to player1. After the first round of the battle, player1 has 1400 hp remained and player2 has 1700 hp remained.

Since player1 and player2 are both alive, the battle goes to the next round. If one player has less than or equal to 0 hp remained, then this player is dead, and its opponent wins the game. Please note that the dp for all players will always be the same.

The following is a summary of both the initial settings of players and type of attacks in the game.

- Inital Settings for Each Player
  2000 hp (health points), 500 dp (defense points)

- Type of Attacks

  1. Attack with Sword
     attack power = 800 hp
     damage to enemy = attack power - defense of enemy

  2. Use Black Magic
     attack power = 600 hp
     extra attack: 40% chance to invoke another sword attack with power 1000 hp
     damage to enemy = attack power - defense of enemy + damage from extra attack.

The basic logic of the game is implemented in "Battle.pl", which has been provided to you. Please check the "Battle.pl" for more details. Arathorn is instantiated as a human player, which needs to accept input from the command line regarding which attack to use in each round. Lapras is instantiated as an AI player. You need to ensure equal chance to invoke a sword attack or black magic attack for the AI Player. In each round, Arathorn always attacks first and then Lapras attacks. The battle ends when one of them dies.

3.4 Perl Classes

Please follow the classes `Player`, `HumanPlayer` and `AIPlayer` defined below in your Perl implementation. You are free to add other variables, methods or classes. We would start the program by running: `Battle.pl`

1. **Class** `Player`

   A super class representing player in the game. You have to implement it with the following components:

   - **Instance Variable(s)**
     `name`
     - This is the name of the player.

`hp`

- This is a variable representing hp of the player.

`dp`

- This is a variable representing dp of the player.

- **Instance Method(s)**

  `new(name)`

  - Instantiate an object of Class `Player` with its name, and return this object.

  `attackWithSword(enemy)`

  - Attack enemy with sword.

  `useBlackMagic(enemy)`

  - Attack enemy using black magic.

  `isDead`

  - Return 0 if hp of the player is larger than 0, otherwise return 1.

  `displayInfo`

  - Display hp and dp of the player.

2. **Class** `AIPlayer`

   This class extends class `Player`. You have to implement it with the following components:

   - **Instance Method(s)**

     `attack(enemy)`

     - Attack enemy using either sword or black magic, which should be decided by computer.

3. **Class** `HumanPlayer`

   This class extends class `Player`. You have to implement it with the following components:

   - **Instance Method(s)**

     `attack(enemy)`

     - Attack enemy using either sword or black magic, which should be decided by human. In this function, you need to accept input from the command line. A valid input is either 's' or 'b', where 's' means attacking with sword and 'b' means attacking with black magic. If an invalid input is detected, you need to request the user to input again until you get a valid one.

3.5 Execution Context

We provide you with the main program, "Battle.pl". Your implementation should be able to run with this main program.

3.6 Use of Random Numbers

In your program, there should only be two places, where random numbers are needed. First is in function `useBlackMagic` of class `Player` to invoke another sword attack with probability of 0.4. Following is a template showing you how to call random number generator (rand()) in function `useBlackMagic`. You should *strictly follow* our template to call rand().

```
function useBlackMagic
    ...
    my $rand = rand()
    if $rand < 0.4 then
        invoke another sword attack with power 1000 hp.
        ...
    end if
    ...
end function
```

The second place, where random number is needed, is in function `attack` of class `AIPlayer`, to invoke either a sword attack or a black magic attack with equal probability, 0.5. Following is a template showing you how to call random number generator (rand()). You should *strictly follow* our template to call rand().

```
function attack
    ...
    my $rand = rand()
    if $rand < 0.5 then
        invoke a sword attack.
        ...
    else
        use black magic.
        ...
    end if
    ...
end function
```

Except these two places stated above, you should not call random number generator anywhere in your program. In the main program, we set the seed of the random number generator to be 10. By running your program with the same settings, we should get the same output. Please do not set the seed of the random number generator by yourself.

3.7 Input/Output Specification

You should input and output in the command line.

- **Input**
  Input from the command line is needed only in function `attack` of class `HumanPlayer`, where 's' means attacking with sword and 'b' means using black magic.
- **Output**
  The main program has already output very important information starting with character '#' for our grading script to use. So in the rest of the implementation, you can output any information as you want as long as it does not contain character '#'.

Please do not input or output anything containing character '#', since it is reserved for our marking script to use. Please note that you can not modify "Battle.pl". And the implementation of each class need to be in a separate file with the file name being exactly the class name.

3.8 Grading Criteria

We will design several test cases to test your program. In each test case, we will use a set of inputs to execute your program. After the execution of your program, we will extract those lines starting with '#' and compare them with ours. So please *strictly follow* our design in input, output and calling random number generator.

3.9 A Running Example of Our Demo Program

Figure 2 in the supplementary materials shows a running example of our demo program.

4 Report

You simple report should answer the following question within ONE A4 pages.

Using codes in Task 2, state where you use dynamic scoping and why you use it here. If you need to implement it using static scoping instead of dynamic scoping, what will you do? You do not need to implement it, just describe your idea clearly. Now try to summarize your understanding about dynamic scoping.

5 Submission Guidelines

Please read the guidelines CAREFULLY. If you fail to meet the deadline because of submission problem on your side, marks will still be deducted. So please start your work early!

1. In the following, **SUPPOSE**

   your name is *Chan Tai Man*,
   your student ID is *1155234567*,
   your username is *tmchan*, and
   your email address is *tmchan@cse.cuhk.edu.hk*.

2. In your source files, insert the following header. REMEMBER to insert the header according to the comment rule of Perl.

```
/*
 * CSCI3180 Principles of Programming Languages
 *
 * --- Declaration ---
 *
 * I declare that the assignment here submitted is original except for source
 * material explicitly acknowledged.  I also acknowledge that I am aware of
 * University policy and regulations on honesty in academic work, and of the
 * disciplinary guidelines and procedures applicable to breaches of such policy
 * and regulations, as contained in the website
 * http://www.cuhk.edu.hk/policy/academichonesty/
 *
 * Assignment 3
 * Name : Chan Tai Man
 * Student ID : 1155234567
 * Email Addr : tmchan@cse.cuhk.edu.hk
 */
```

   The sample file header is available at

   `http://course.cse.cuhk.edu.hk/~csci3180/resource/header.txt`

3. Late submission policy: less 20% for 1 day late and less 50% for 2 days late. We shall not accept submissions more than 2 days after the deadline.

4. The report should be submitted to VeriGuide, which will generate a submission receipt. The report should be named "report.pdf". The VeriGuide receipt of report should be named "receipt.pdf". The report and receipt should be submitted together with codes in the same ZIP archive.

5. `Tar` your source files to `username.tar`
   First, `tar` all your source files for task 1 and task 2 into task1.tar and task2.tar respectively.

```
tar cvf task1.tar Blackjack.pl Deck.pm Participant.pm Dealer.pm \
Player.pm HumanPlayer.pm AIPlayer.pm <otherFiles>
tar cvf task2.tar Battle.pl Player.pm AIPlayer.pm HumanPlayer.pm <otherFiles>
```

Then, `tar` task1.tar, task2.tar, report.pdf and receipt.pdf to `username.tar`.

```
tar cvf tmchan.tar task1.tar task2.tar report.pdf receipt.pdf
```

6. `Gzip` the `tarred` file to `username.tar.gz` by

```
gzip tmchan.tar
```

7. `Uuencode` the `gzipped` file and send it to the course account with the email title "HW3 *studentID yourName*" by

```
uuencode tmchan.tar.gz tmchan.tar.gz \
| mailx -s "HW3 1155234567 Chan Tai Man" csci3180@cse.cuhk.edu.hk
```

8. Please submit your assignment using your Unix accounts.
9. An acknowledgement email will be sent to you if your assignment is received. **DO NOT** delete or modify the acknowledgement email. You should contact your TAs for help if you do not receive the acknowledgement email within 5 minutes after your submission. **DO NOT** re-submit just because you do not receive the acknowledgement email.
10. You can check your submission status at

    `http://course.cse.cuhk.edu.hk/~csci3180/submit/hw3.html`.

11. You can re-submit your assignment, but we will only grade the latest submission.
12. Enjoy your work :>

Supplementary Materials

```
WELLCOME, BLACKJACK PLAYER!
Please input some infor to initialize the game.
*****Number of players:*****
2
Player 1:
Player type: 'h' human player, 'a' AIPlayer
h
Name of player : Sandy
Player 2:
Player type: 'h' human player, 'a' AIPlayer
a
Name of player : Mary
*****Only one dealer*****
Name of dealer : Alice
Configuration complete!

*****Game start*****
's' to start, 'q' to quit
s
Deal two cards to all players
Cards in all players' hand.
Player Sandy
A, 2
Total value: 3(hard hand) with 1 Ace.
Player Mary
4, K
Total value: 14 with 0 Ace.
Dealer Alice
3, J
Total value: 13 with 0 Ace.

*****Player Sandy's turn*****
'h' to hit, 's' to stand
h
Sandy received a card 5.
Cards in hand: A, 2, 5
Total value: 8(hard hand) with 1 Ace.
'h' to hit, 's' to stand
s
Sandy chooses to stand.
Cards in hand: A, 2, 5
Total value: 8(hard hand) with 1 Ace.

*****Player Mary's turn*****
Mary received a card 2.
Cards in hand: 4, K, 2
Total value: 16 with 0 Ace.
Mary received a card 4.
Cards in hand: 4, K, 2, 4
Total value: 20 with 0 Ace.
Mary chooses to stand.
Cards in hand: 4, K, 2, 4
Total value: 20 with 0 Ace.

*****Dealer Alice's turn*****
Alice received a card J.
Cards in hand: 3, J, J
Total value: 23 with 0 Ace.
Dealer Alice is busting.
3, J, J
Total value: 23 with 0 Ace.
#Sandy A 2 5
#Mary 4 K 2 4
#Alice 3 J J
Winner between the dealer and each player.
#Sandy Mary
's' to start, 'q' to quit
q
```

Figure 1: A Running Example of Blackjack

```
*****BATTLE START*****
#Arathorn hp: 2000 dp: 500
#Lapras hp: 2000 dp: 500
*****Round 1*****
Arathorn's turn.
's' to attack with sword (power: 800 hp)
'b' to use black magic (power: 600 hp), extra effect:  40% chance to invoke another sword attack with power 1000 hp.
b
Arathorn uses black magic to Lapras.
What a pity!
Arathorn is not lucky enough to invoke another sword attack.
Lapras's turn.
Lapras uses black magic to Arathorn.
What a pity!
Lapras is not lucky enough to invoke another sword attack.
*****After Round 1*****
#Arathorn hp: 1900 dp: 500
#Lapras hp: 1900 dp: 500
*****Round 2*****
Arathorn's turn.
's' to attack with sword (power: 800 hp)
'b' to use black magic (power: 600 hp), extra effect:  40% chance to invoke another sword attack with power 1000 hp.
s
Arathorn attacks Lapras with sword.
Lapras's turn.
Lapras uses black magic to Arathorn.
What a pity!
Lapras is not lucky enough to invoke another sword attack.
*****After Round 2*****
#Arathorn hp: 1800 dp: 500
#Lapras hp: 1600 dp: 500
*****Round 3*****
Arathorn's turn.
's' to attack with sword (power: 800 hp)
'b' to use black magic (power: 600 hp), extra effect:  40% chance to invoke another sword attack with power 1000 hp.
b
Arathorn uses black magic to Lapras.
What a pity!
Arathorn is not lucky enough to invoke another sword attack.
Lapras's turn.
Lapras uses black magic to Arathorn.
What a pity!
Lapras is not lucky enough to invoke another sword attack.
*****After Round 3*****
#Arathorn hp: 1700 dp: 500
#Lapras hp: 1500 dp: 500
*****Round 4*****
Arathorn's turn.
's' to attack with sword (power: 800 hp)
'b' to use black magic (power: 600 hp), extra effect:  40% chance to invoke another sword attack with power 1000 hp.
b
Arathorn uses black magic to Lapras.
What a pity!
Arathorn is not lucky enough to invoke another sword attack.
Lapras's turn.
Lapras uses black magic to Arathorn.
What a pity!
Lapras is not lucky enough to invoke another sword attack.
*****After Round 4*****
#Arathorn hp: 1600 dp: 500
#Lapras hp: 1400 dp: 500
*****Round 5*****
Arathorn's turn.
's' to attack with sword (power: 800 hp)
'b' to use black magic (power: 600 hp), extra effect:  40% chance to invoke another sword attack with power 1000 hp.
s
Arathorn attacks Lapras with sword.
Lapras's turn.
Lapras uses black magic to Arathorn.
What a pity!
Lapras is not lucky enough to invoke another sword attack.
*****After Round 5*****
#Arathorn hp: 1500 dp: 500
#Lapras hp: 1100 dp: 500
*****Round 6*****
Arathorn's turn.
's' to attack with sword (power: 800 hp)
'b' to use black magic (power: 600 hp), extra effect:  40% chance to invoke another sword attack with power 1000 hp.
b
Arathorn uses black magic to Lapras.
Black Magic!
Arathorn successfully invokes another sword attack this round with power 1000 hp.
Lapras's turn.
Lapras uses black magic to Arathorn.
What a pity!
Lapras is not lucky enough to invoke another sword attack.
*****After Round 6*****
#Arathorn hp: 1400 dp: 500
#Lapras hp: 500 dp: 500
*****Round 7*****
Arathorn's turn.
's' to attack with sword (power: 800 hp)
'b' to use black magic (power: 600 hp), extra effect:  40% chance to invoke another sword attack with power 1000 hp.
b
Arathorn uses black magic to Lapras.
Black Magic!
Arathorn successfully invokes another sword attack this round with power 1000 hp.
*****Arathorn wins!*****
#Arathorn
```

Figure 2: A Running Example of "Defend Your Kingdom"