On the 10000 Yen Game [*]

Koji Nakano
Department of Information Engineering, Hiroshima University

1 Introduction

Many programming competitions have been held to improve programming abilities of students. However, they may be too difficult for students to understand the rules, and may require to use complicated data structures. Hence the students may be discouraged to join the programming competitions.

The main contribution of this article is to introduce *the 10000 Yen Game*, which has a quite simple rule and does not require complicated data structures and programming techniques. This game was originally designed for a programming competition. Students who join this programming competition need to write a C-language program to decide the betting values in each round of the game. Since the rule and the programming interface are so simple that students who have just started to learn C-language programming can join the programming competition. Also, although the rule is very simple, this game is so deep. A lot of algorithmic and programming techniques must be used to develop a strong an good program.

2 The 10000 Yen Game

The 10000 Yen Game is a two-player game. Initially, two players A and B have 10000 yen each. Each player bet totally 10000 yen in 10 rounds. In each of the 10 rounds, a player bet smaller wins both. If the two player bet the same, each player get the money he/she bet. Let a_i and b_i $(1 \leq i \leq 10)$ the values that players A and B bet in the i-th round. We can write the game formally as follows:

[*]This document is trasnrated from the article *"Koji Nakano, On the 1000 Yen Game, The Bulletin of the LA Symposium, Vol 52, pp. 23–27, January 2009"* written in Japanese.

- The total betting value of each player must be 10000, that is,

$$\sum_{i=1}^{10} a_i = \sum_{i=1}^{10} b_i = 10000.$$

- The betting value is at least 0, that is, in each round i $(1 \leq i \leq 10)$, both $a_i \geq 0$ and $b_i \geq 0$ hold.
- A player bet smaller win both. In other words, If $a_i < b_i$, then A win $a_i + b_i$ yen. If $a_i > b_i$, then B win $a_i + b_i$ yen. If $a_i = b_i$, then A and B get a_i $(= b_i)$ yen each.

Note that, if a bet larger wins both, the game is trivial. It is also trivial, if the game has only two rounds.

We assume that, when players A and B decide betting values a_i and b_i, they know the history of the betting values $a_1, \ldots, a_{i-1}$ and $b_1, \ldots, b_{i-1}$, and these values can be used for deciding the next betting values a_i and b_i. Clearly, at the end of the 10-th round, A and B have 20000 yen totally. The winner of this *game* is a player who won larger total values. This game is repeated 1000 times for two players A and B. Players A and B can use the history of the past games. For example, player A can analyzes the strategy of B based on the history, and can adjust the betting value in each round of each game. If the number of won games of A is larger than that of B, then A is the winner of this *match*.

In our laboratory, more than 30 students joined the competition, and we select the winner of them by the round-robin contest.

In the programming competition, students must write C-language player programs which decide the betting values based on the history of the game. We have developed a server program that manages the two player programs. Let A and B also denote two player programs. The server program execute player programs A and B as child processes using folk and exec system calls. The betting values are transfered through the pipes as follows: The player programs A and B write a_1 and b_1, respectively, to the standard output using the "printf" library call. The sever program receives them through the pipes, and displays the result of the first round. After that, the server program send b_1 and a_1 to The player programs A and B, respectively, though the pipe. The player programs A and B receives these values from the standard input using the "scanf" library call. This protocol is repeated for 10 rounds. The number of games of the match is given as the first argument (i.e `argv[1]`) of player programs. The 10-round game is repeated for `argv[1]` times.

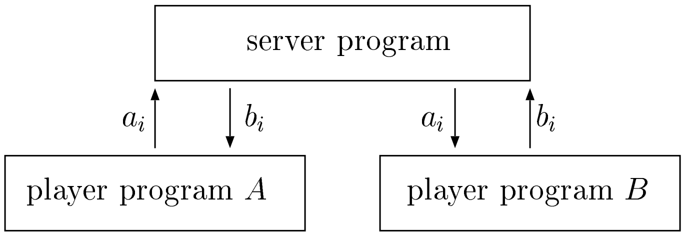

Figure 1: The server program and player programs

3 Examples of player programs for the 10000 yen game

This section shows three simple player programs.

always1000.c: This player program bets 1000 yen in every round.

minus1.c: This player program bets 0 yen in the first round. From the 2nd to 9th rounds, it bets the opponent betting value of the previous round minus 1. Finally it bets the remaining value in the 10-th round.

random.c: From 1st to 9th rounds, the betting value is selected uniformly at random from 0 to the remaining value. The remaining value is bet in 10-th round.

Lists 1,2, and 3 are the program lists. The definitions for including header files are omitted.

List 1: always1000.c

```
int main(int argc,char **argv){
  int n,j,k,num_game;
  num_game=atoi(argv[1]);
  for(k=1;k<=num_game;++k)
    for(j=1;j<=10;++j){
      printf("1000\n");
      fflush(stdout);
      scanf("%d",&n);
    }
  exit(0);
}
```

List 2: minus1.c

```
int main(int argc,char **argv){
  int n,j,k,num_game;
  num_game=atoi(argv[1]);
  for(k=1;k<=num_game;++k){
    int bet,sum=0;
    printf("0\n"); // round 1
    fflush(stdout);
    scanf("%d",&n);
    for(j=2;j<=9;++j){ //rounds 2-9
      bet=(n>0?n-1:0);
      printf("%d\n",bet);
      fflush(stdout);
      sum+=bet;
      scanf("%d",&n);
    }
    printf("%d\n",10000-sum); // round 10
    fflush(stdout);
    sum+=bet;
    scanf("%d",&n);
  }
  exit(0);
}
```

Let us see the results of the matches between `always1` and `minus1`. For simplicity, we assume 1-game match. List 4 is the output of the server program. The list shows that alwasy1000.c win 3008 yen, minus1.c 16992 yen, and thus minus1.c is a winner of the game. Intuitively, one of the good strategy is to select the betting value of a round should be a bit smaller than the opponent betting value. Since random.c uses works randomly, it may win or lose for any opponent. With high probability, random.c looses against always1000.c and wins agings minus1.c. Thus, these three games are in a three-cornered tie.

4 Various strategy for the 10000 yen game

Clearly, in the 10000 yen game, it is not possible for one of the two players to win all the 10 rounds. Each player should have both win rounds and loose rounds. If we can bet a bit smaller value than the opponent betting value when the opponent bet a large value, we win the large value. Thus, one of the good strategy is to guess a round in which the opponent bets a large value. If we can guess such round and the value, we can bet a bit smaller value in that stage and get the sum of the large betting values. For this

List 3: random.c

```
int main(int argc,char **argv){
  int n,i,j,num_game;
  num_game=atoi(argv[1]);
  srand(time(NULL));
  for(j=1;j<=num_game;++j){
    int bet,remain=10000;
    for(i=1;i<=9;++i){
      bet=random()%remain;
      printf("%d\n",bet); // rounds 1-9
      fflush(stdout);
      remain-=bet;
      scanf("%d",&n);
    }
    printf("%d\n",remain); // round 10
    fflush(stdout);
    scanf("%d",&n);
  }
  exit(0);
}
```

purpose, our player program need to guess the opponent strategy in earlier games of the 1000-game matches. Let us show several strategies developed by students in the contest.

- In the first round, we bet a large value, say, 7000 yen. We bet the remaining value almost equally but randomly in the remaining rounds. With high probability, we loose in the first round. For example, if the opponent bet 1000 yen, then we loose 8000 yen. After the first round, we have 3000 yen while the opponent has 9000 yen. In the remaining rounds, we bet expected 333 yen per round, while the opponent must bet expected 1000 yen per round. So, with good probability, we win in all the remaining rounds. If we win in the remaining 9 rounds, we win 12000 yen.
- We memorize the opponent betting values of each round in the previous game. We bet a bit smaller value of each corresponding round in the first 9 rounds. If the opponent bet a similar value, we win in the first 9 rounds.
- We implement several strategies including above two, we select one of the strategies at random for each round. We record the winning percentage of each strategy, and select a strategy such that better winning percentage strategies are selected with higher probability.

List 4: The result of the match always1 vs. minus1

```
always1000 vs minus1
 1: 1000 [ 0] −> 0 1000
 2: 1000 [ 999] −> 0 2999
 3: 1000 [ 999] −> 0 4998
 4: 1000 [ 999] −> 0 6997
 5: 1000 [ 999] −> 0 8996
 6: 1000 [ 999] −> 0 10995
 7: 1000 [ 999] −> 0 12994
 8: 1000 [ 999] −> 0 14993
 9: 1000 [ 999] −> 0 16992
10: [1000] 2008 −> 3008 16992
     3008 [16992]
minus1 won.
```

It seems to me that there exists the best strategy. An strategy A_1, it is always possible to develop a strategy A_2 which beats A_1. In general, we can develop a strategy A_{i+1} which beats A_i, and thus, we can develop ultimate strategy A_n for some large n. However, it is not guaranteed that A_n beats all of the $n-1$ strategies $A_1, A_2, \ldots A_{n-1}$. It may be possible that there exists three particular strategies A, B, and C such that no strategy beats all of these three strategies.

5 Conclusions

In this article, we have introduces the 10000 yen game, which is simple but deep. Even a beginner of C-programming language can develop a player program. The reader can download the source programs including the server program and several player programs from

`http://www.cs.hiroshima-u.ac.jp/~nakano/10000/`.