CS101 Computer Programming and Utilization

Milind Sohoni

June 13, 2006

The story so far ...

- functions
- file handling
- structs
- Srirang's problem
- Classes

This week...

Another real-life problem

A Game

- Ramu and Shamu, both want to enter IIT. Both are reasonably equally prepared.
- The benefit of getting into IIT is 100.
- The price of Kota coaching is 25.
- If one of them does Kota and the other doesnt then the Kota chap gets in.
- If both do Kota, things are equal again.

A Game

- Ramu and Shamu, both want to enter IIT. Both are reasonably equally prepared.
- The benefit of getting into IIT is 100.
- The price of Kota coaching is 25.
- If one of them does Kota and the other doesnt then the Kota chap gets in.
- If both do Kota, things are equal again.

The Questions

- What is the best strategy for Ramu and Shamu?
- Can they discover it?

A Game

- Ramu and Shamu, both want to enter IIT. Both are reasonably equally prepared.
- The benefit of getting into IIT is 100.
- The price of Kota coaching is 25.
- If one of them does Kota and the other doesnt then the Kota chap gets in.
- If both do Kota, things are equal again.

The Questions

- What is the best strategy for Ramu and Shamu?
- Can they discover it?

This data can be summarized as follows:

- Each player has two options, viz., 0 (Home) and 1 (Kota).
- Each player chooses a play, and then observes the pay-off.

Pay-Off 1

	0	1
0	50	0
1	100	50

Pay-Off 2

	0	1
0	50	100
1	0	50

The game again

Just so that we understand this:

- If both play 0, then both have an equal chance of getting into IIT. Thus the expected payoff for each is 50.
- If player1 plays 1 and player2 a 0, then player 1 gets 100-25=75. Player2 gets nothing.
- If both play 1, then they are equal again, and each has an expected gain of 50-25=25.

Pay-Off 1

	0	1
0	50	0
1	100	50

Pay-Off 2

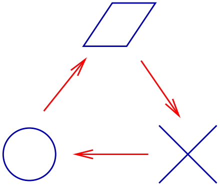
	0	1
0	50	100
1	0	50

The costs are as follows

Option	player1	player2
0	0	0
1	25	25

Another Game

Here is a common game.

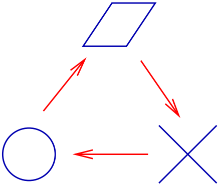


- This is a common game. Essentially 2 beats 1, 1 beats 0 but 0 beats 2.
- If both players play the same, then no one wins/loses.

Again, the same question: How should you play this game?

Another Game

Here is a common game.



- This is a common game. Essentially 2 beats 1, 1 beats 0 but 0 beats 2.
- If both players play the same, then no one wins/loses.

Again, the same question: How should you play this game?

Payoff 1

	0	1	2
0	0	0	1
1	1	0	0
2	0	1	0

Payoff 2

	0	1	2
0	0	1	0
1	0	0	1
2	1	0	0

Each move costs uniformly 1 unit.

games as a class

What should this class contain?

- The number of options for each player?
- The pay-off matrices.
- A procedure to read the payoff matrices.
- A procedure to return the payoffs for each player, once a play is made.

games as a class

What should this class contain?

- The number of options for each player?
- The pay-off matrices.
- A procedure to read the payoff matrices.
- A procedure to return the payoffs for each player, once a play is made.

Lets make a class called game as follows:

```
class game
{
 private:
   int payoff1[8][8],
             payoff2[8][8];
 public:
  void ReadIn(void);
// reads two payoff matrices
 int options1,options2;
 void payoffs(int op1, int op2,
            int& p1 , int& p2);
// computes the payoffs and
// returns them in p1 and p2
};
```

games1.cpp

Lets make a class called game as follows:

```
class game
{
 private:
   int payoff1[8][8],
             payoff2[8][8];
 public:
  void ReadIn(void);
// reads two payoff matrices
 int options1,options2;
 void payoffs(int op1, int op2,
            int& p1 , int& p2);
// computes the payoffs and
// returns them in p1 and p2
};
```

games1.cpp

Lets make a class called game as follows:

```
class game
{
 private:
   int payoff1[8][8],
             payoff2[8][8];
 public:
  void ReadIn(void);
// reads two payoff matrices
 int options1,options2;
 void payoffs(int op1, int op2,
            int& p1 , int& p2);
// computes the payoffs and
// returns them in p1 and p2
};
```

```
void game::ReadIn(void)
{
   int i,j;
   cin>> options1 >> options2;
   for (i=0;i<options1;i=i+1)
    for (j=0;j<options2;j=j+1)
           cin >> payoff1[i][j];
   for (i=0;i<options1;i=i+1)
    for (j=0;j<options2;j=j+1)
           cin >> payoff2[i][j];

   return;
}
void game::payoffs(int op1,
     int op2, int& p1 , int& p2)
{
  p1=payoff1[op1][op2];
  p2=payoff2[op1][op2];
  return;
}
```

The main program

```
int main()
{
  game g; int o1,o2,p1,p2;
  g.ReadIn();
  cout << g.options1 << " " <<
               g.options2 << "\n";
  g.payoffs(0,0,p1,p2);
  cout << p1  << " "<< p2 << "\n";
  g.payoffs(1,1,p1,p2);
  cout << p1  << " "<< p2 << "\n";
  g.payoffs(1,0,p1,p2);
  cout << p1  << " "<< p2 << "\n";
}
```

A sample main program:

- g.ReadIn initializaes the game:
  - reads number of options for each player
  - reads in the two pay-off matrices.
- Next, there are some sample plays. Note that p1,p2 were called by reference.

More meaningfull main programs SOON.

The main program

```
int main()
{
  game g; int o1,o2,p1,p2;
  g.ReadIn();
  cout << g.options1 << " " <<
               g.options2 << "\n";
  g.payoffs(0,0,p1,p2);
  cout << p1  << " "<< p2 << "\n";
  g.payoffs(1,1,p1,p2);
  cout << p1  << " "<< p2 << "\n";
  g.payoffs(1,0,p1,p2);
  cout << p1  << " "<< p2 << "\n";
}
```

```
2 2

50 0
100 50

50 100
0 50
```

```
[sohoni]$ ./a.out <kota
2 2
50 50
50 50
100 0
```

The Players class

Lets make a class for the players as well. What must this class store?

- The number of options the player has.
- Surely, the cost of each option.
- A strategy to make a play!
- total costs, total benefits.

The Players class

Lets make a class for the players as well. What must this class store?

- The number of options the player has.
- Surely, the cost of each option.
- A strategy to make a play!
- total costs, total benefits.

```
class player1
{
  private:
    int options, costs[8];
    int count, sumcost, sumpay;
  public
    void init(int)
    // initializes player
    // reads in costs
    int play(void);
    // makes a play
    void returns(int);
    // accepts the payoffs;
    void report(void);
    // prints a summary
}
```

The Players class

```
class player1
{
  private:
    int options, costs[8];
    int count, sumcost, sumpay;
  public
    void init(int)
    // initializes player
    // reads in costs
    int play(void);
    // makes a play
    void returns(int);
    // accepts the payoffs;
    void report(void);
    // prints a summary
}
```

- options is the number of options this player has.
- costs stores the cost of executing each option.
- count,sumcost,sumpay stores aggregates.
- report will produce a summary of the transactions of this player.

player2 is similar.

The main program

```
int main()
{
  game g; int o1,o2,p1,p2;
  player1 pp1,pp2; int i,N=300;

  g.ReadIn();
  pp1.init(g.options1);
  pp2.init(g.options2);
  for (i=0;i<300;i=i+1)
  {
   o1=pp1.play();
   o2=pp2.play();
   g.payoffs(o1,o2,p1,p2);
   pp1.returns(p1);
   pp2.returns(p2);
  };
  pp1.report();
  pp2.report();
}
```

- A game g and player1 pp1,pp2 are declared. The number of trials is set to 300.
- g.ReadIn() causes the payoff matrices to be loaded and g.options1 and g.options2 to be set.
- The next two statements initializes pp1 and pp2.

The main program

```
int main()
{
  game g; int o1,o2,p1,p2;
  player1 pp1,pp2; int i,N=300;

  g.ReadIn();
  pp1.init(g.options1);
  pp2.init(g.options2);
  for (i=0;i<300;i=i+1)
  {
   o1=pp1.play();
   o2=pp2.play();
   g.payoffs(o1,o2,p1,p2);
   pp1.returns(p1);
   pp2.returns(p2);
  };
  pp1.report();
  pp2.report();
}
```

- A game g and player1 pp1,pp2 are declared. The number of trials is set to 300.
- g.ReadIn() causes the payoff matrices to be loaded and g.options1 and g.options2 to be set.
- The next two statements initializes pp1 and pp2.
- Inside the for loop:
  - Each player plays and receives a payoff.
- Finally, a report is prepared.

games2.cpp-player1

Lets see what the functions in player1 look like:

```
void player1::init(int N)
{
  int i;
  options=N; count=0;
  sumcost=0; sumpay=0;
  for (i=0;i<N;i=i+1)
     cin >> costs[i];
  return;
}
int player1::play(void)
{
  int d;
  d=rand()%options;
  count=count+1;
  sumcost=sumcost+costs[d];
  return(d);
}
```

games2.cpp-player1

Lets see what the functions in player1 look like:

```
void player1::init(int N)
{
  int i;
  options=N; count=0;
  sumcost=0; sumpay=0;
  for (i=0;i<N;i=i+1)
     cin >> costs[i];
  return;
}
int player1::play(void)
{
  int d;
  d=rand()%options;
  count=count+1;
  sumcost=sumcost+costs[d];
  return(d);
}
```

- player1.init tells this player how many options it has. It then initializes all constants to zero.
- It also reads in the costs of each option.
- player1.play calls a C++ function called rand() which returns a random large integer between 0 and RAND_MAX.
- This random number modulo the number of options is decided as the next play.
- sumcost is updated.

more player1

```
void player1::returns(int x)
{
  sumpay=sumpay+x;
  return;
}

void player1::report(void)
{
  float avgcost, avgpay;
  avgcost=1.0*sumcost/count;
  avgpay=1.0*sumpay/count;
  cout << "number " << count << "\n";
  cout << "avg cost " << avgcost << "\n";
  cout << "avg payoff " << avgpay << "\n";
  return;
}
```

These functions are simple enough!

- player1.returns merely updates the payoffs so far.
- player1.report produces a report.

The main program

```
int main()
{
  game g; int o1,o2,p1,p2;
  player1 pp1,pp2; int i,N=300;

  srand(time(NULL));
  g.ReadIn();
  pp1.init(g.options1);
  pp2.init(g.options2);
  for (i=0;i<300;i=i+1)
  {
   o1=pp1.play();
   o2=pp2.play();
   g.payoffs(o1,o2,p1,p2);
   pp1.returns(p1);
   pp2.returns(p2);
  };
  pp1.report();
  pp2.report();
}
```

- Everything is now SET
- The game is initialized.
- The players get their options and costs.
- The play begins for 300 games.
- Current players play randomly. Better players soon

Better players soon!

Whats the output?

```
2 2

50 0
100 50

50 100
0 50

0 25 // cost of options
0 25 // for each player
```

```
[sohoni]$ ./a.out <kota2
number 300
avg cost 11.5833
avg payoff 50.1667
*****
number 300
avg cost 11.5
avg payoff 49.8333
```

- Player 1 played 300 rounds with an average cost of 11.58 and an average return of 50.16 making a net gain of 38.58.
- Similarly, Player 2 made an average gain of 38.34.

Whats the output?

```
2 2

50 0
100 50

50 100
0 50

0 25 // cost of options
0 25 // for each player
```

```
[sohoni]$ ./a.out <kota2
number 300
avg cost 11.5833
avg payoff 50.1667
*****
number 300
avg cost 11.5
avg payoff 49.8333
```

- Player 1 played 300 rounds with an average cost of 11.58 and an average return of 50.16 making a net gain of 38.58.
- Similarly, Player 2 made an average gain of 38.34.

Do these numbers makes sense?

- expected costs=0.5*0+0.5*25=12.5! looks OK.
- expected earnings=0.25*50 +0.25*0+0.25*100 +0.25*50=50! OK again.

Whats the output?

```
2 2

50 0
100 50

50 100
0 50

0 25 // cost of options
0 25 // for each player
```

```
[sohoni]$ ./a.out <kota2
number 300
avg cost 11.5833
avg payoff 50.1667
*****
number 300
avg cost 11.5
avg payoff 49.8333
```

What does all this mean?

- Player 1 played 300 rounds with an average cost of 11.58 and an average return of 50.16 making a net gain of 38.58.
- Similarly, Player 2 made an average gain of 38.34.

KOTA made a tidy profit of 23.08 × 300.

Whats the output?

```
2 2

50 0
100 50

50 100
0 50

0 25 // cost of options
0 25 // for each player
```

```
[sohoni]$ ./a.out <kota2
number 300
avg cost 11.5833
avg payoff 50.1667
*****
number 300
avg cost 11.5
avg payoff 49.8333
```

What does all this mean?

- Player 1 played 300 rounds with an average cost of 11.58 and an average return of 50.16 making a net gain of 38.58.
- Similarly, Player 2 made an average gain of 38.34.

KOTA made a tidy profit of 23.08 × 300.

Is there a better STRATEGY?

A strategy

The player should maintain an average of her earnings for each option. The next play should be based on this information.

A strategy

The player should maintain an average of her earnings for each option. The next play should be based on this information.

- The class should include variables for maintaining this information.
- The `player.play` procedure should use the above data.
- The `player.returns` procedure should update this data.
- There should be sufficient randomness so that the player doesnt get too conditioned by initial few outputs

Another strategy: games3.cpp

```
class player1
{
  private:
    int last, options, costs[8];
    float probs[8], wts[8];
    int count, sumcost, sumpay;
  public:
    ...
};
```

```
void player1::init(int N)
{
  int i; last=0; ...
  for (i=0;i<N;i=i+1)
  {
     cin >> costs[i];
     wts[i]=10;
  };
  return;
}
```

Lets explain:

- Each player stores her cumulative profits for every option.
- Her next play is based on the above data.
    - The more profit in that option, the more is the chance of playing that option.

Another strategy: games3.cpp

```
class player1
{
  private:
    int last, options, costs[8];
    float probs[8], wts[8];
    int count, sumcost, sumpay;
  public:
    ...
};
```

```
void player1::init(int N)
{
  int i; last=0; ...
  for (i=0;i<N;i=i+1)
  {
     cin >> costs[i];
     wts[i]=10;
  };
  return;
}
```

Lets explain:

- Each player stores her cumulative profits for every option.
- Her next play is based on the above data.
  - The more profit in that option, the more is the chance of playing that option.
- wts will store the cumulative earnings per option initilized to 10.
- probs will store the probability of playing that option.
- last stores the move made last.

player1.play

```
int player1::play(void)
{ ...
  d=rand();
  rr=1.0*d/RAND_MAX;
  count=count+1;
  sum=0;
  for (i=0;i<options;i=i+1)
    sum=sum+wts[i];
  for (i=0;i<options;i=i+1)
    probs[i]=wts[i]/sum;

  now op is found

  last=op;
  sumcost=sumcost+costs[op];
  return (op);
}
```

Whats happening:

- rr is a random number between 0 and 1.
- If wts[0]=300 and wts[1]=400, then probs[0]=3/7 and probs[1]=4/7.
- If $0 \leq rr \leq probs[0]$ then op=0, else op=1.
- Next, the total costs are updated, and last is stored, to be used later.

player1.returns

```
void player1::returns(int x)
{
  sumpay=sumpay+x;
  wts[last]=wts[last]
         +x-1*costs[last];
  return;
}
```

Recall that our last move is stored in last. Now that the returns are x, we must update our statistics. This is simple:

- sumpay is updated.
- Now last is used to update the profits wts[last].
  - This is clearly old profits + x-cost_of_last_move.

what do we get?

```
[sohoni]$ ./a.out <kota2
number 1000
avg cost 2.1
avg payoff 4.75
*****
number 1000
avg cost 24.725
avg payoff 95.25
```

```
[sohoni]$ ./a.out <kota2
number 1000
avg cost 2.725
avg payoff 49.75
*****
number 1000
avg cost 2.85
avg payoff 50.25
```

Whats happening:

- We have shown TWO runs. Note that because of randomness, one player may learn something quite different from another run.

what do we get?

```
[sohoni]$ ./a.out <kota2
number 1000
avg cost 2.1
avg payoff 4.75
*****
number 1000
avg cost 24.725
avg payoff 95.25
```

```
[sohoni]$ ./a.out <kota2
number 1000
avg cost 2.725
avg payoff 49.75
*****
number 1000
avg cost 2.85
avg payoff 50.25
```

Whats happening:

- We have shown TWO runs. Note that because of randomness, one player may learn something quite different from another run.
- In the first run, pp1 tunes out and pp2 goes to KOTA.
- In the next run, both player boycott KOTA!

KUCH KUCH HOTA HAI

Whats next?

Assignment

- Is there a better and reasonable strategy for the two players to discover the [0,0] best strategy? Note that you cannot see what the other player has played.
- What if you knew what the other joker has played?
- Try out your strategies for other games.
- What is the programming changes if the two players wanted to play different strategies?