iSnake - Intelligent Multiplayer Snake
http://isnake.sourceforge.net

Submitted to
Open Software Competition
LOCUS 2007/08

May 08, 2008

Abhishek Dutta  Jitendra Harlalka  Suraj Sapkota
adutta.np@gmail.com  jitendra.harlalka@gmail.com  ssapkota@gmail.com

Institute of Engineering, Pulchowk Campus
Lalitpur, Nepal
# Table of Contents

1 Objectives

2 Introduction
   2.1 Definitions ......................................................... 2
   2.2 Components of iSnake ........................................... 3

3 iSnake Client Application
   3.1 Client Encoder/Decoder ........................................ 4
   3.2 Client Network Interface ....................................... 4
   3.3 Input Handler ..................................................... 4
   3.4 Game Field Matrix ............................................... 5
   3.5 Game Field Canvas ............................................... 5
      3.5.1 Anatomy of snakes drawn on the game field ............ 5
      3.5.2 Performance of Game Field Canvas ....................... 6
   3.6 User Interface Components ..................................... 6
   3.7 Game Controller ................................................ 6

4 iSnake Autonomous Intelligent Opponent Application ....... 8
   4.1 Performance of the two path finding algorithms .......... 8

5 iSnake Game Server
   5.1 Server Encoder/Decoder ........................................ 10
   5.2 Server Network Interface ....................................... 10
   5.3 Virtual Game Field .............................................. 10
   5.4 Player Information Manager ................................... 11
   5.5 Random Number Pool ............................................ 11
   5.6 Status Server .................................................... 11
   5.7 Server Core ....................................................... 11
   5.8 Performance of Game Server ................................... 11
      5.8.1 Performance evaluation in Local Network ............ 12
      5.8.2 Performance evaluation in Homogeneous LAN ........ 12
      5.8.3 Performance evaluation in Heterogeneous LAN ....... 15

6 iSnake Game Server Manager at sourceforge.net ............ 18

7 Program Flow ......................................................... 19

8 Project Management and Documentation ....................... 23

9 Market Viability .................................................... 25

10 Limitations .......................................................... 26
List of Figures

2.1 Major components of iSnake project ........................................... 3
3.1 Block diagram of iSnake Client Application ................................. 4
3.2 Encoding of direction keys by Input handler module ...................... 5
3.3 Anatomy of a game field snake .................................................. 6
3.4 Game Field Canvas render time for 3 players ............................... 7
4.1 Block diagram of iSnake Autonomous Intelligent Opponent Application . 8
4.2 Performance of the path finding algorithms: (a) Path length plot and (b) Turn around time plot for the two path finding algorithms .................. 9
5.1 Block diagram of iSnake game server ........................................... 10
5.2 Response time plot for 2 players in local network ......................... 13
5.3 Response time plot for 3 players in local network ......................... 13
5.4 Response time plot for 4 players in local network ......................... 14
5.5 Response time plot for 2 players in Homogeneous LAN ................... 14
5.6 Response time plot for 3 players in Homogeneous LAN ................... 15
5.7 Response time plot for 2 players in Heterogeneous LAN .................. 16
5.8 Response time plot for 3 players in Heterogeneous LAN .................. 17
6.1 Block diagram of iSnake Game Server Manager (GSM) .................... 18
7.1 iSnake game application enters in STAGE-1 and exits through STAGE-5 .... 19
7.2 iSnake game application starts in STAGE-1 and this stage involves connection to game server specified by the player .............................. 20
7.3 In STAGE-2 players can chat with each other. The game will start only when all the players connected to the game server send READY signal .............. 20
7.4 STAGE-3 involves receiving game data (like wall coordinates, food coordinate, snakes start position, etc) .......................................... 21
7.5 STAGE-4 involves synchronization with the game server .................. 22
7.6 STAGE-5 represents the state when game is being played ................ 22
8.1 Subversion Activity Statistics for iSnake ...................................... 23
8.2 Screen shot of a snippet of source code showing a method comments conforming to javadoc standards ................................................. 24
A.1 Bounding rectangle and paths formed by it .................................. 31
A.2 Four possible paths generated by source and target partial bounding rectangles (spbr,tpbr - depicted using red and green dotted lines respectively) for a given hopping point ......................................................... 33
A.3 Empty game field with source (snake) and target(food) placed at 'S' and 'T' respectively ................................................................. 33
A.4 Procedure of finding path for a given (S,T) pair - Simple Case ........... 35
Acknowledgment

We offer our gratitude to project supervisor Asst. Prof. Jayaram Timsina (Deputy Head of Department, Dept. Of Electronics and Computer Engineering, Pulchowk Campus) for guiding us throughout the development of this project.

We acknowledge the creators of Quantum Random Bit Generator Service - QRBG [http://random.irb.hr], which provides our game server with high quality true random numbers.

We would like to thank all those forum members at javagaming.org and Sun Java Forum [http://forum.java.sun.com] who replied to our endless queries without any complain. We also acknowledge Alok Singh, Bibek Shrestha, Binit Amatya, Lakesh Kansakar, Lilasing Pun, Sankalp Labh for their support during testing and profiling of iSnake.
Abstract

iSnake aims to bring the fun and simplicity of snake game (popular among cell phone users) with some new features. It includes a computer controlled intelligent opponent whose aim is to challenge the human players during the game. It also has the multiplayer feature that allows more than one players to play a game over a network. The iSnake project explores a new dimension in the traditional snake game to make it more interesting and challenging. The simplicity of this game makes it an ideal candidate for our research on multiplayer functionality and artificial intelligence in games.
1. Objectives

This game aims to change the way people think of traditional snake game. It will offer the experience of commercial multiplayer games to the player retaining the simplicity of traditional snake game. The main objectives of this project are:

- Create a snake game that will have all the functionality of traditional snake games.
- Introduce multiplayer functionality in the game that will allow several players to play a game simultaneously. It should be able to give the experience of a real time multiplayer game to the players.
- Introduce computer controlled intelligent opponent (unique feature of this game) to make the game more challenging and interesting. The movement and action of intelligent opponents will be controlled by computer whose aim will be to eat the food before human players capture it.
2. Introduction

Snake game is quite popular among mobile phone users. This game was introduced in early Nokia cell phones and has been tagged as an addictive game since then. In those traditional snake game, the game was quite straight forward and simple. Hence, players lost interest after playing it for some time.

This project aims to explore some new dimensions in snake game while retaining the fun and simplicity associated with traditional snake games. iSnake includes autonomous intelligent opponent snake that will challenge the human players during the game and multiplayer functionality that allows more than one player to play a game over a network.

Computer controlled intelligent snakes, whose aim is to eat the food before the other players, appear in the game field. Two path finding algorithms viz: Blackmamba and Viper have been developed to embed intelligence into these snakes. The feature of computer controlled autonomous intelligent opponent snakes has not been spotted in any variants of Snake game and is hence unique feature of iSnake.

2.1 Definitions

The definition of important terms used in this document are give below:

**Remote Player** It refers to the player who plays the iSnake game over a network.

**iSnake Client Application** It is an application that is used to play the iSnake game by remote players.

**iSnake Game Server** It is the server that hosts iSnake game. Remote players use iSnake Client Application to join game hosted by the game server.

**iSnake Autonomous Intelligent Opponent Application** It refers to the application that joins a game server as a player named ‘Blackmamba’ or ‘Viper’ and plays the game autonomously. The movement of these snakes is controlled by the path finding algorithms Blackmamba and Viper.

**iSnake Game Server Manager (GSM) at sourceforge.net** It refers to the web application hosted at http://isnake.sourceforge.net that keeps track of iSnake game servers active on the internet.

**Homogeneous LAN** It refers to network comprising of Local Area Network of a single ISP.

**Heterogeneous LAN** It refers to the network comprising of Local Area Network of two or more ISP.

**Cell** A cell, in context of Game Field Canvas, refers to the 10x10 pixel area in the Game Field Canvas referred using its x and y cell coordinate. A game field canvas contains 58x58 cells.

**Response Time** It is the time elapsed between the instant of a player pressing an arrow key and the instant at which the player observes the actual turning of the snake. This parameter is used to evaluate the performance of iSnake game server.
2.2 Components of iSnake

iSnake consists of four components that work together to provide the experience of a complete online multiplayer game. The integration between these four components is illustrated in Figure 2.1

![Figure 2.1: Major components of iSnake project](image)

We will discuss all these components in detail in next sections. A brief description of each component is given below:

**iSnake Client Application** It allows remote players to join a game server and play the snake game hosted on that server. It also provides chat messaging functionality among the players active in the game.

**iSnake Autonomous Intelligent Opponent Application** This application is a clone of iSnake client application with an exception that the InputHandler (which is attached to the keyboard of local player in iSnake client application) is replaced by a module that implements the path finding algorithm. This module joins a game server just like a remote human player and plays the game to challenge the human players during the game.

**iSnake Game Server** Game server is based on Client Server Multiplayer game architecture. It is used to host an iSnake game which can be joined and played by remote players over a network. It maintains the state of the game which is transmitted to each remote player in every game cycle.

**iSnake Game Server Manager (GSM) at sourceforge.net** iSnake GSM at sourceforge.net is used to manage information about all the iSnake Game servers hosted on the Internet. It acts as a central repository of the active game servers that can be joined by any remote player.
3. iSnake Client Application

Remote players use iSnake Client Application to join a iSnake game server. The block diagram is given in Figure 3.1

![Block diagram of iSnake Client Application](image)

Figure 3.1: Block diagram of iSnake Client Application

We describe each components of the block diagram now.

3.1 Client Encoder/Decoder

This module performs the encoding and decoding of messages leaving/arriving the client network interface module using the protocol standard described in Appendix C.

The communication packets are encoded in byte format for the purpose of transmission and are decoded in similar fashion. This allows us to transmit minimum number of bytes required for communication. The encoding and decoding of packets in iSnake Client Application is handled by `net.sf.isnake.codec.clientEncoder` and `net.sf.isnake.codec.clientDecoder` modules in the iSnake source code tree respectively.

3.2 Client Network Interface

It provides an interface to the Game Controller module for communication with the game server hosted at local/remote computer. It is responsible for triggering of appropriate methods of Game Controller when message from game server is received.

3.3 Input Handler

It manages the task of sampling key strokes from local player and forwarding it to Game Controller when requested. It maintains a queue of size 2 so that quick keystrokes are not lost. It is active only when the game is in running mode. The encoding of direction keys (Right, Down, Left, Up) is done using two byte variables dx and dy as shown in Figure 3.2. Hence for each change in direction of the snake, we have to transmit just 2 bytes of data. **When no keystroke is pressed,**
no data is transmitted and the game server assumes that no change in direction of the snake has occurred.

### 3.4 Game Field Matrix

Game Controller maintains the complete state of the game using game field matrix. It is a 2D array of size 58x58 (equal to the dimension of game field). Each game field object has a unique identifier in the game field. Game Controller updates the cells of this matrix in each cycle to register the changes that occur in the game.

### 3.5 Game Field Canvas

It represents the game field as seen by the player. Game Controller analyzes the game field matrix in each game cycle and updates the game field canvas to represent the state of game in that game cycle. Double Buffering [using `java.awt.Canvas.createBufferStrategy()`] has been implemented to avoid flickering of game field. Each block in the game field has dimension 10x10 pixels.

The update of game field canvas occurs in the way similar to the refreshing technique of a cathode ray monitor. The game field matrix and game field canvas are updated in separate thread. The update of game field canvas starts by scanning each column of the 1st row in the game field matrix, then 2nd row and so on up to the 58th row.

#### 3.5.1 Anatomy of snakes drawn on the game field

The game field canvas contains 58x58 cells each of dimension 10x10 pixels. The snake of a player is drawn using several predefined images representing different parts of the snake as illustrated in Figure 3.3. When a player turns its snake (using arrow keys), the orientation of Head0 is determined. This orientation is reflected to the remaining parts of the snake moving one cell towards the tail in each game cycle. The logical structure of snake is maintained by Game Field Matrix which is updated in each game cycle. Each part of the snake has a unique identifier which is used to draw the graphical representation of that part on the game field.

The 10x10 pixels images of the snake’s body parts are loaded as accelerated image when the game application is started. Hence we were able to limit the rendering time of the game field under 100ms (game cycle time) barrier.

---

1 refer Section 2.1 for definition
3.5.2 Performance of Game Field Canvas

The time taken to render the game field has been used as a parameter to examine the performance of Game Field Canvas module. The plot in Figure 3.4 shows the render time when three players are playing the game in heterogeneous LAN.

From the above plot we observe that the rendering time does not cross the 100ms game cycle barrier for any of the remote players. The Game Field Canvas is supposed to complete the task of rendering the game field within the game cycle time of 100ms. For more number of players (say 8 players) the Game Field Canvas may, occasionally, breach the 100ms game cycle barrier. The effect of this breach will be that the rendering of one game cycle will be skipped. As the game cycle is very small (100ms), this effect will not usually be noticed by the players.

The variation in rendering time for different players can be attributed to the difference in platforms (CPU, memory, etc and the operating system - OS) of the remote players. The performance of Java Runtime Environment (JRE) is known to vary with OS.

3.6 User Interface Components

This module includes all the components, except game field, visible to the player. The look and feel of default swing components have been overridden to give the feel of a game to the players. MIT OCW’s course “[6.831] User Interface Design and Implementation” was referred extensively during design of most of the user interface components of this game.

We adopted the Iterative Design technique for the development of user interface. Members of the iSnake team and some volunteers were the source of feedback during the UI design phase.

3.7 Game Controller

It is the most important component of the iSnake client application. It coordinates the working of all the other modules in the application and handles all the messages received from the game server. Game Controller maintains the game cycle when the game is running.

The Game Controller performs the following task in each game cycle (of 100ms)
Figure 3.4: Game Field Canvas render time for 3 players

- If keystroke of local player is available, transmit it to the game server. If no keystroke is available for this cycle, do not transmit any information. *When no information is received about a player the game server assumes no change in direction for that player’s snake.*

- Receive the dx_dy multicast packet from the game server
  - check if any player has collided with wall (info. present in the multicast packet)
  - check if any player has eaten the food (info. present in the multicast packet)
  - if none of the above two is true, then update the state of game in Game Field Matrix

- Render the game field using the information in Game Field Matrix

This game cycle is synchronized with the game cycle time of the game server. All the updates to game field matrix are done during this cycle time. After expiry of each game cycle the game field is repainted to reflect the changes in the game field.
4. iSnake Autonomous Intelligent Opponent Application

This application is a clone of iSnake Client application with an exception that the InputHandler module (present in iSnake Client Application) is replaced by the module that implements path finding algorithm for autonomous intelligent opponent snakes.

These are computer controlled snakes, in the game, whose aim is to challenge the human players. We have two implementations of path finding algorithms to create intelligent autonomous opponent snakes. A detailed paper describing the algorithm used by these two implementation is given in Appendix A and Appendix B. These algorithms return the shortest possible path from given source (S) and target (T) coordinate pair considering the obstacles (if any) present in the game field. The code name\(^1\) for these two implementations are:

- **Blackmamba** refer to Appendix A
- **Viper** refer to Appendix B

4.1 Performance of the two path finding algorithms

The following test methodologies were used to judge the performance of the two path finding algorithms viz. Blackmamba and Viper.

- For a given wall, we choose 8 coordinate pairs for source and target (S,T) located at strategic points so that the process of path computation is complex.

- We defined a timeout value for path computation as 250ms to prevent long computation cycles. The path finding algorithms were designed to return a negative value for path length and turn around time if the computation time exceeded the timeout value.

\(^1\) named after two popular species of venomous snakes
A path is formed by a collection of continuous cell coordinates. The number of cell coordinates in a path was taken as the path length.

The time elapsed during path computation was calculated using System.nanoTime()²

We developed a simple JUnit test to gather data (path length and turn around time) related to the performance of these two algorithms and the results were plotted using gnuplot as shown in Figure 4.2.

![Path Length](image1.png) ![Turn around time](image2.png)

(a) Path length plot  (b) Turn around time plot

Figure 4.2: Performance of the path finding algorithms: (a) Path length plot and (b) Turn around time plot for the two path finding algorithms

**Turn around time plot** The time elapsed between the instant of supplying the (source,target) coordinate pair (S,T) to the algorithm and the instant when it returns a path for supplied (S,T) pair is called turn around time. From the plot of Figure 4.2b it is clear that the turn around time for Viper is always smaller as compared to Blackmamba.

**Path Length plot** The length of path (computed by counting the number of cell coordinates in the path) returned by the two path finding algorithms is depicted by the plot of Figure 4.2a. It is clear from the plot that Viper implementation results in smaller paths (and hence efficient) as compared to Blackmamba implementation.

The negative value of turn around time and path length for the coordinate pairs 5,7,8 suggests that the algorithm, Blackmamba, was not able to compute a path for the specified (S,T) pair in given timeout period (250 ms for this test). Any further tests regarding the performance of two path finding algorithms was not required because Blackmamba implementation could not compute path for some of the supplied (S,T) coordinate pair.

²“This method can only be used to measure elapsed time ... “ - API specification of java.lang.System.nanoTime() for version 6 of the Java Platform, Standard Edition
5. iSnake Game Server

iSnake game server handles the multiplayer feature of this game and allows multiple iSnake client applications to play the game hosted by that particular game server. The main components of iSnake game server are shown in Figure 5.1.

5.1 Server Encoder/Decoder

This module performs the encoding and decoding of messages leaving/arriving the server network interface module using the protocol standard described in Appendix C.

5.2 Server Network Interface

It provides an interface to the Server Core module for communication with the remote/local players of the game being hosted. It is responsible for triggering of appropriate methods of Server Core when message from remote/local players is received.

5.3 Virtual Game Field

Game server maintains the state of the game using a 2D array of size 58x58 (similar to that used by Game Controller refer to 3.7). It is maintained by the game server to check whether the food has been eaten and whether any player has collide with the wall. It maintains the head coordinate (not the coordinates for tails) of each player. The head coordinate of players is moved in each game cycle and checked for the presence of wall/food in that coordinate position. Server Core generates corresponding event (collide or food eaten) and all the active players are informed about the event in the same cycle.

Figure 5.1: Block diagram of iSnake game server
5.4 Player Information Manager

It manages the information about the players involved in the game. Player information like name, location, score, snake’s starting position, snake color, etc are maintained. A new entry is added whenever a new player joins the game. Similarly, when the player leaves the game, it is removed.

5.5 Random Number Pool

Game server maintains a buffer of random numbers obtained from Quantum Random Bit Generator service [QRBG - http://random.irb.hr]. This gives the game server access to true random number. These random data is used to generate the position of food and the starting coordinate of each player’s snake.

If the random number service is unreachable pseudo random numbers are generated using java.util.Random class provided by Java.

5.6 Status Server

Status server maintains all the information required to reply the current status of the game server. The service of status server is utilized by iSnake Game Server Manager(iSnake - GSM). The response of status server is a well formed XML document that is parsed by iSnake GSM to display information about the game server in the website.

A typical response of Status Server is given below:

```xml
<?xml version=1.0 encoding=UTF-8?>
<iSnake>
  <GameServerData>
    <GameServerAddress>124.41.228.219</GameServerAddress>
    <GameServerPort>9669</GameServerPort>
    <NoOfPlayersOnline>5</GameServerLocation>
    <GameServerStatus>Waiting</GameServerStatus>
  </GameServerData>
</iSnake>
```

5.7 Server Core

It is the most important component of the iSnake Game Server. It coordinates the working of all the other modules in the application and handles all the messages received from the remote players. When the game is in ”Waiting” mode, Server Core provides the facility of chat messaging to the game players. During this state new players can join the game.

When all the players have sent signal to start the game, the Server Core changes state to Running mode. If a new player tries to join the game, it receives a NAK response. In this mode, Server Core maintains a game cycle time during which it receives the movement coordinates (in terms of deltaX and deltaY) from the players. If a player does not send any packet during this cycle time, server considers the movement coordinate sent in last cycle for the current game cycle. After expiry of cycle time, game server broadcasts a packet containing movement coordinates of each player to all the players active in the game. The game server also checks if any player has eaten the food or if any players have collided to the wall in each cycle.

5.8 Performance of Game Server

In ideal scenario, the response time for each key press event should be zero. The factors that contribute to non-zero response time are:

1 refer to 6iSnake Game Server Manager at sourceforge.net
the delay in packet transmission introduced by Local Area Network.

• the load (number of remote players connected to the game server) on the game server

The following test methodologies were used to judge the performance of iSnake game server.

• Response time\(^2\) of players was used as a parameter to evaluate the performance of iSnake game server.

• The test sessions were performed on Local LAN, Homogeneous LAN and Heterogeneous LAN with the help of some volunteers.

• Automatic log aggregator, especially developed for this purpose, was used to collect log data from remote player’s iSnake Client application. This reduced the chances of error in data collection. Gnuplot was then used to plot the charts from these collected data.

• The time elapsed during path computation was calculated using System.nanoTime()\(^3\)

NOTE: The Internet Service Providers chosen for performance evaluation in homogeneous and heterogeneous LAN were:

**wlink** WorldLink Communication (http://www.wlink.com.np)

**netplus** Netplus Technology Pvt. Ltd (http://www.netplus.com.np)

**ioe-cit** Internet service provided by Center of Information Technology [Pulchowk Campus, Lalitpur, Nepal] to IOE New Boys Hostel. (http://cit.ioe.edu.np)

### 5.8.1 Performance evaluation in Local Network

We used a standard 8 port Switch to connect four PC in a network. Test data were collected for the case when 2 players, 3 players and 4 players were playing a game. The resulting plot is shown in Figure 5.2, 5.3, 5.4

After observing the three plots in Figure 5.2, 5.3, 5.4 we conclude that:

• the response time is usually below 200ms. Hence this delay is not easily noticed by the players. Such response time does not cause any noticeable effect during game play.

• increase in number of players connected to the game server does not contribute much to the response time.

### 5.8.2 Performance evaluation in Homogeneous LAN

Subscribers of Worlink Cable internet service were chosen for this test. The plot of response time is shown in Figure 5.5, 5.6.

After observing the two plots in Figure 5.5, 5.6 we conclude that:

• the plots have nature similar to the plots for Local Network.

---

\(^2\) refer to section 2.1

\(^3\) “This method can only be used to measure elapsed time ... “- API specification of java.lang.System.nanoTime() for version 6 of the Java Platform, Standard Edition
Figure 5.2: Response time plot for 2 players in local network

Figure 5.3: Response time plot for 3 players in local network
Figure 5.4: Response time plot for 4 players in local network

Figure 5.5: Response time plot for 2 players in Homogeneous LAN
5.8.3 Performance evaluation in Heterogeneous LAN

Subscribers of Worlink Cable internet service and Netplus cable internet service were chosen for this test. The plot of response time is shown in Figure 5.7, 5.8.

After observing the two plots in Figure 5.7, 5.8 we conclude that:

- The response time is very high for some remote player (player: binit, lilasing pun). Remote players easily observe this high response time.

- The remote players which are in homogeneous LAN, with respect to the game server, still have small response time (usually <200ms) as depicted by Figure 5.8. These players do not observe any effect of the very high response time of other remote player (forming a part of heterogeneous network). **This effect is due an important aspect of iSnake game server design.** The game server enforces the following policy during formation of DxDy multicast packet - *If movement coordinates (dx,dy) of a remote player is not available for the game cycle, assume it to be 0 (ie: dx=0, dy=0).* Because of this policy, the game server does not wait for the arrival of packets from remote client (having very high response time). Hence, the players that have small response time can play the game without any side effect of a high response time remote player.

- iSnake game, in its present form, cannot be played comfortably by remote clients forming a heterogeneous network.
Figure 5.7: Response time plot for 2 players in Heterogeneous LAN
Figure 5.8: Response time plot for 3 players in Heterogeneous LAN
iSnake Game Server Manager (iSnake GSM) hosted at http://isnake.sf.net is used to manage all the information about iSnake game servers being hosted over the Internet.

The iSnake GSM has been developed using PHP. Java Web Start technology has been used to deploy iSnake application at our website which automatically downloads/installs iSnake application and its library dependencies. This provides the gamers with the facility of one click launch of the iSnake application. It also make the distribution of updates of the iSnake application to the end users very convenient.

The iSnake application deployed at our website has been digitally signed by the iSnake team to address the security issues related to launch of Internet applications.
7. Program Flow

The program flow of iSnake Client application has State based model. The GameController\textsuperscript{1} enforces strict policy on the execution of methods by ClientNetworkInterface\textsuperscript{2} module. GameController maintains the current stage of application at all the time during the application’s life cycle. Methods registered for that stage can only be invoked (by other appropriate modules). The stage is changed on the corresponding request from the predefined module.

The stages involved in the life cycle of iSnake Client application is illustrated in Figure 7.2 to Figure 7.6. Similar state based model is also followed by the iSnake Game server.

![Diagram](image)

Figure 7.1: iSnake game application enters in STAGE-1 and exits through STAGE-5

\textsuperscript{1}Section 3.7
\textsuperscript{2}Section 3.2
Figure 7.2: iSnake game application starts in STAGE-1 and this stage involves connection to game server specified by the player.

Figure 7.3: In STAGE-2 players can chat with each other. The game will start only when all the players connected to the game server send READY signal.
Figure 7.4: STAGE-3 involves receiving game data (like wall coordinates, food coordinate, snakes start position, etc)
Figure 7.5: STAGE-4 involves synchronization with the game server

Figure 7.6: STAGE-5 represents the state when game is being played
8. Project Management and Documentation

iSnake is a project registered at sourceforge.net\(^1\) (SF). Most of project management resources and tools available at SF were used during the design and development phase of this project.

The SVN (subversion - a version control system) service provided by SF was used to manage the source code of iSnake. Subversion Activity Statistics for iSnake project (generated on 2008-05-06) is shown in Figure 8.1.

![Subversion Activity Statistics for iSnake](image)

**Figure 8.1: Subversion Activity Statistics for iSnake**

We collaborated on project documents (including prototype designs, project plan, TODO list, etc) using WIKI (http://isnake.wiki.sourceforge.net).

JUnit tests were developed to independently test some of the modules before integration. The integration of modules developed by the three developers was performed in three phases:

- Phase 1 Integration (Sep. 28, 2007) - chat functionality of the game was tested successfully
- Phase 2 Integration (Oct. 06, 2007) - successful testing of basic version of multiplayer snake game
- Phase 3 Integration (Feb 17 - 26, 2008) integration of all the modules for iSnake 0.1 Beta release

Documentation of every task being done in the project was a priority for all the team members. Almost every portion of the source code contains full code documentation conforming to Javadoc standards. There exists two path finding algorithms that implements intelligent opponent in

\(^1\)project website: http://www.sourceforge.net/projects/isnake
Figure 8.2: Screen shot of a snippet of source code showing a method comments conforming to javadoc standards.

```java
/**
 * Returns the object present in specified cell of game field
 * @param x x-coordinate of the specified cell
 * @param y y-coordinate of the specified cell
 * @return a Short value representing the object in cell specified by (x,y)
 */
public Short getGameFieldObject(int x, int y) {
    return fieldMatrix[x][y];
}
```

the game viz-Blackmamba and Viper. These two algorithms have been fully documented with illustrations in Appendix A and Appendix B. The protocol devised for communication between game server and clients has been documented in Appendix C.
9. Market Viability

On-line gaming is a booming industry. As iSnake has been deployed using Java Web Start technology it can be easily used by commercial gaming portals like Zapak.com. Moreover, if ported to Mobile phone platform, it can be deployed as revenue generating (as the client applications would use connectivity services like GPRS, CDMA, etc) add on service by mobile phone operators.

The introduction of high speed ADSL Internet introduced by Nepal Telecom has created exciting new possibilities for on-line multiplayer games like iSnake in Nepal.
10. Limitations

iSnake has the following limitations:

- The present implementation of iSnake can only be played in Local Network and Homogeneous LAN. Due to response time it cannot be played in Heterogeneous LAN and on the Internet.

- Path finding algorithms (Blackmamba and Viper) implemented in this game have their own computation limitations which has been describe in Appendix A, B.

- iSnake’s Game Server Manager (iSnake GSM) located at http://isnake.sf.net is still in its early development phase. There are some unresolved security issues.

- Full stress testing (with 8 players in all forms of network connection) of the application has not been done yet. Hence, the response of iSnake in such environment cannot be predicted.
11. Future Enhancements

- iSnake can be ported to One Laptop Per Child - OLPC (which uses Sugar Desktop environment) platform which contains WIFI connectivity option that can allow OLPC users to play this game with their friends. Local WIFI network formed by kids using OLPC laptops can be used as a platform for iSnake’s deployment.

- The presence of several connectivity options (Bluetooth, WIFI, GPRS, CDMA) in cell phones makes it a very attractive platform for a multiplayer game like iSnake. As iSnake is written in Java it can be ported to cell phones that support Java application.

- The game can be made more attractive by giving a 3D look to the game field.

- Further optimization and design changes in the iSnake game server can be done to make it fit for playing in heterogeneous network and internet.
12. Conclusion

We were successful in creating a multiplayer version of traditional snake game. The computer controlled intelligent opponents have been successfully tested in the game and is a unique feature of iSnake.

iSnake project provided us with a great platform for research on multiplayer functionality and Artificial Intelligent in games.

We learned several project management techniques also used by professionals to develop large scale project. The experience of working in team and integration of modules developed independently, with just requirement specifications, is a very important achievement for the iSnake team.

Plots obtained after profiling of iSnake application gave us a good insight regarding its performance. This will help us further improve the performance of iSnake application and make it fit for playing in a heterogeneous network and over the internet.
13. References

Books

1. The Java Programming Language, by Ken Arnold, James Gosling, David Holmes


   Ronald L. Rivest, Clifford Stein


Websites


2. The Inkscape Tutorials Blog - http://inkscapetutorials.wordpress.com


5. Timing Framework - https://timingframework.dev.java.net/


7. MIT Open Course Ware - 6.831 User Interface Design and Implementation

8. Amit’s A* Page Tutorial on A* Path finding algorithms
   http://theory.stanford.edu/~amitp/GameProgramming/


10. JavaDesktop Community Blog - http://community.java.net/javadesktop/


    http://www.curious-creature.org/

14. A Distributed Multiplayer Game Server System

15. The Java Developers Almanac 1.4 - http://www.exampledepot.com/


18. Quantum Random Bit Generator Service (QRBG) - http://random.irb.hr/
A. Blackmamba path finding algorithm

Proposed by: Abhishek Dutta

A.1 Some Definitions and Notations used in this paper

Bounding Rectangle (br) It is a rectangle in which the source and the target lie at two opposite corners of the rectangle. Source Bounding Rectangle (sbr0) and Target Bounding Rectangle (tbr0) represent the bounding rectangles from source and target which are the same rectangle.

Partial Bounding Rectangle (pbr) These rectangles touch either the source or the target node and not both. Two partial bounding rectangles touching 'S' and 'T' form paths from source to the target node.

Source Partial Bounding Rectangle (spbr) The pbr touching the source node 'S'.
Target Partial Bounding Rectangle (tpbr) The pbr touching the target node 'T'.

![Bounding rectangle and paths formed by it](image)

Figure A.1: Bounding rectangle and paths formed by it

**brNM** denotes a bounding rectangle that spans (or touches) N nodes along the x-axis and M nodes along the y-axis.

**spbrNM** denotes a source partial bounding rectangle that spans (or touches) N nodes along the x-axis and M nodes along the y-axis.

**tpbrNM** denotes a target partial bounding rectangle that spans (or touches) N nodes along the x-axis and M nodes along the y-axis.
hpNM denotes a hopping point (coordinate specified with reference to source) along which spbrNM and tpbrNM are formed. This coordinate point is common to both spbrNM and tpbrNM.

**Numbering of Paths** For any given bounding rectangle (br) there are only two possible paths as show in Figure A.1b. Path numbering convention that has been applied is:

**Path1** The x-coordinate of the path first changes followed by change in y-coordinate

**Path2** The y-coordinate of the path first changes followed by change in x-coordinate

A source (S) and target (T) can be placed in four possible ways such that they lie on opposite ends of the diagonal. Considering the four position (A,B,C,D) as depicted in Figure A.2, we have the following four cases:

- **'S' placed at position A and 'T' placed at position C**
  Path1 = ABC, Path2 = ADC
- **'S' placed at position B and 'T' placed at position D**
  Path1 = BAD, Path2 = BCD
- **'S' placed at position C and 'T' placed at position A**
  Path1 = CDA, Path2 = CBA
- **'S' placed at position D and 'T' placed at position B**
  Path1 = DCB, Path2 = DAB

A.2 **Basis of the algorithm**

For any given br, if there exists a path from source to target inside the br then it will be the shortest path. In absence of any obstacles, the cost of moving from one node to another is constant throughout the game field. Hence, the shortest path is along the bounding rectangle edges if no obstacles are present in the path as shown in Figure A.1b. Both path1 and path2 have equal traveling cost provided that no obstacles are present along those paths.

A.3 **Paths generated by partial bounding rectangles**

Four paths are generated by the source and target partial bounding rectangles for the given hopping point as shown in Figure A.2. For any given (S,T) pair, we move the hopping point along the diagonal joining S and T. Special considerations is required when:

- obstacles (wall) are present along the paths formed using the bounding and partial bounding rectangles.
- no possible path can be found using all the possible combinations of the partial bounding rectangles.
- the bounding rectangle is not a square (ie: for brNM, N M)

A.4 **Description of the algorithm**

Instead of considering all the possible cases at once, let us consider the working of algorithm in several stages (with increasing level of complications).
A.4.1 Symmetric Bounding Rectangles (Simple Case)

For illustration purpose, let us first consider a game field of 6x6 nodes where source and target are placed at 'S' and 'T' respectively as shown in Figure A.3. The following series of steps are executed to find a path from source to target. If obstacle is found in all possible path of an stage the next stage is checked. The series of steps taken to obtain a path from 'S' to 'T' are:

**Procedure of finding path for a given (S,T) pair**

1. Figure A.4a: initially the hopping point is hp66 (on the target). two paths formed along the bounding rectangle br66 (shown in red dotted rectangle in Figure A.4a) are checked for presence of obstacles. If no obstacles are found, either path1 or path2 are chosen (refer to Figure A.1b). As both path1 and path2 have same path length, choice between these two paths is made based on some combination of sub paths and choosing the path whose length is shortest.

2. Figure A.4b: four paths (refer to Figure A.2) formed along the source and target partial bounding rectangle (spbr55, tpbr22) are checked for presence of obstacle.

3. Figure A.4c: four paths (refer to Figure A.2) formed along the source and target partial bounding rectangle (tpsbr44, ptbr33) are checked for presence of obstacle.
4. Figure A.4d: four paths (refer to Figure A.2) formed along the source and target partial bounding rectangle (tpsbr33, ptbr44) are checked for presence of obstacle.

5. Figure A.4e: four paths (refer to Figure A.2) formed along the source and target partial bounding rectangle (tpsbr22, ptbr55) are checked for presence of obstacle.

6. Figure A.4f: This step is not necessary to execute as the task has already been performed in Step 1 using the bounding rectangle (br66).

Path computation overhead calculation
Let the time required to check whether a partial bounding rectangle has an obstacle = T1
time required to check whether a bounding rectangle has an obstacle = T2
for a game field NxN (row x col)
also let (to create the worst case)
• the source 'S' be located at (1,1)
• the target 'T' be located at (N,N)

Total no. of possible hopping points (excluding the source) = N  1
Hence, total time required to calculate a path1 = T2 + N * T1

Backtracking
If the six steps of Figure A.4 does not give a path from 'S' to 'T' the technique of backtracking will be applied. To illustrate the process of backtracking let us consider the scenario shown in Figure A.5a.

The paths generated from the bounding rectangle (br44) and all the possible partial bounding rectangles contain obstacle. Hence the process discussed above will not result in any path from 'S0' to 'T'. For such scenario we can apply the process of backtracking.

Backtracking involves moving to the next outer (as inner bounding rectangles do not contain any path for sure) bounding rectangle a path to the target is found. Figure A.5b shows the result of backtracking. Backtracking from S0 to S1 (the next outer bounding rectangle) results in a bounding rectangle br55. This bounding rectangle does not also result in any path from S1 to T. Hence, next outer bounding rectangle is checked.

The next outer bounding rectangle (br66) is formed at node S2 as shown in Figure A.5c. This bounding rectangle has a path from S2 to T. Hence, the possible path is calculated to reach from S2 to T.

As we now have a path from S2 to T, we need to calculate a path from S0 to S2 so that we can ultimately reach T from S0. Now we calculate the possible path from S0 to S2 using the steps discussed above as shown in Figure A.5d.

We will apply the algorithm, discussed in next section, to merge the subpaths formed during backtracking to form a single continuous path from S0 to T.

Merging the paths formed while backtracking
While backtracking several sub paths are created while requires to be merged to form the final path from source to target. Let us consider a case as depicted in Figure A.6. Due to the presence of wall (green blocks), S and T has to backtrack to S' and T' positions respectively. Here we obtain three sub paths which are:

**Path from S to S’** : depicted by red colored line from S to S’ (SubPath1)

**Path from S’ to T** : depicted by blue colored line from T to T’ (SubPath2)
(a) STEP1: The two paths of source partial bounding rectangle (spbr66, shown in red) is checked for presence of obstacles (wall)

(b) STEP2: Four possible paths formed from the target partial bounding rectangle (tpbr22, shown in green) is checked along with spbr55 for presence of obstacle.

(c) STEP3: Four possible paths formed from tpbr33 and spbr44 are checked for presence of obstacle.

(d) STEP4: Four possible paths formed from tpbr44 and spbr33 are checked for presence of obstacle.

(e) STEP5: Four possible paths formed from tpbr55 and spbr22 are checked for presence of obstacle.

(f) STEP6: Two possible paths formed from tpbr66. This step is not required as it has already been checked in STEP1.

Figure A.4: Procedure of finding path for a given (S,T) pair - Simple Case
(a) Game field scenario when all the paths formed by bounding and partial bounding rectangles contain obstacle

(b) Next outer bounding rectangle (br55) is checked for a path from node S1 to T

(c) Next outer bounding rectangle (br66) is checked for a path from node S2 to T

(d) Path from node S0 to S2 is calculated using the process discussed before (after a path is found from S2 to T)

Figure A.5: Steps involved in backtracking

Path from T’ to T : depicted by pink colored line from T’ to T (SubPath3)

We need to combine these three paths to obtain a final path from S to T (depicted by thick black line). If we add these three paths directly, the final path will contain overlapping paths (which is not required). The following algorithm can be applied to compute the final path (that does not contain any overlapping regions).

START

Step 1: Initialize FinalPath with the contents of SubPath1
        FinalPath = SubPath1

Step 2: Now perform the following operations for each cell coordinate present in SubPath2 and SubPath2 (K=2,3)
        cellCoordinate = SubPathK.getCellCoordinate()
        if(FinalPath contains cellCoordinate) {
            remove cellCoordinate from FinalPath
        } else {
            add cellCoordinate to FinalPath
        }

Step 3: FinalPath contains discontinuities at the bends of path. Search for consecutive cell coordinate for which both x and y values of the coordinate change (ie: |dx| = 1, |dy| = 1).

Step 4: Correct these discontinuities by adding the missing cell coordinates.
Determining next outer bounding rectangle during backtracking

Although the vertex for next outer bounding rectangle (required for backtracking) can be the next coordinate on the extended diagonal (as shown in Figure A.5b, A.5c), a simple technique can greatly reduce the number of backtracking steps of both source and target.

Let source (S) and target (T) be placed as shown in Figure A.7. A path from S to T contains two edges. Path1 contains edge1 and edge2. Path2 contains edge4 and edge3. While determining whether a path contains obstacle, the algorithm must also determine which edges contains obstacles (if any).

We can apply the following algorithm to determine the outer rectangles required during backtracking for source.

```plaintext
if(edge1 contains obstacle) {
    if(edge4 contains obstacle)
        source outer rectangle vertex = S''
    else
        source outer rectangle vertex = S'
}
```
source outer rectangle vertex = \( S' \)

} else {
  if(edge4 contains obstacle)
    source outer rectangle vertex = \( S'' \)
  else
    no backtracking required

}

The outer rectangle for target, during backtracking, can be determined by applying similar logic for edge2 and edge3. This technique works for different orientation of source and target as described in “Numbering of Paths” of Section A.1.

A.4.2 Asymmetric Bounding Rectangles

Simple Case

A problem arises when the bounding rectangle is asymmetric as shown in Figure A.8a. The solution to this problem is to insert required number of dummy nodes between ‘S’ and ‘T’ so that the new bounding rectangle becomes symmetric as shown in Figure A.8b. The algorithm applied in [A] Symmetric Bounding Rectangles (Simple Case) can be now applied by considering the dummy nodes as normal game field nodes free from any obstacles.

![Figure A.8: Asymmetric Bounding Rectangles - Simple case](image)

(a) Case of asymmetric bounding rectangle (br63)  (b) The bounding rectangle (br66) is made symmetric by adding required no. of dummy nodes

S and T lying on same straight line (vertical or horizontal)

This case arises when both ‘S’ and ‘T’ lie on same straight line (horizontal or vertical) as shown in Figure A.9a. The case can be converted to [A] Symmetric Bounding Rectangle (Simple Case) case by adding dummy nodes as shown in Figure A.9b.

A.5 Requirement - Constant time algorithm for obstacle detection

There must exist a constant time algorithm that can answer the question Does the given bounding or partial bounding rectangle contain any obstacle?. The constant time algorithm requirement
Another case of asymmetric bounding rectangle can be converted to “Symmetric Bounding Rectangle (Simple Case)” case by adding dummy nodes.

An important observation for iSnake game is that the obstacles in the game are not dynamic. In other words, the obstacles remain constant until a given FOOD is eaten by one of the players. Hence we can assume that the obstacle is constant during the execution of this path finding algorithm.

We apply the following algorithm to detect whether a given bounding rectangle contains any obstacles.

**START**

**Step 1:** Let \((x_1, y_1)\) = one corner of the bounding rectangle
\((x_2, y_2)\) = diagonally opposite corner of the bounding rectangle
obstacle = a set coordinates defining the obstacle (wall)

**Step 2:** Repeat step 2 for each cell coordinate in obstacle

```java
oc = obstacle.getCellCoordinate()
path1HasObstacle = true, path2HasObstacle = true
if( x >= x1 and x <= x2 ) {
    if(!path1HasObstacle) {
        if( x1y1.getCell().equals(y) )
            path1HasObstacle = true
    }
    if(!path2HasObstacle) {
        if( x2y2.getCell().equals(y) )
            path2HasObstacle = true
    }
}
if ( y >= y1 and y <= y2 ) {
    if(!path1HasObstacle) {
        if( x1y1.getCell().equals(x) )
            path1HasObstacle = true
    }
    if(!path2HasObstacle) {
        if( x2y2.getCell().equals(x) )
            path2HasObstacle = true
    }
}
```

Figure A.9: Asymmetric Bounding Rectangles
if( x1y1.getX().equals(x) )
    path2HasObstacle = true
}
}

Step 3: if(path1HasObstacle)  {
    if(path2HasObstacle)
        return NO PATH POSSIBLE
    else
        return PATH2 is OBSTACLE FREE
} else  {
    if(path2HasObstacle)
        return PATH1 is OBSTACLE FREE
    else
        return PATH1 and PATH2 are OBSTACLE FREE
}

END

A.6 Limitations

1. This algorithm does not consider the transparent game field boundary (entry to one side of the field causes exit in the opposite side of the field as depicted in Figure A.10) during path calculation. Due to this limitation the computed path is not optimal.

2. The hopping points are always taken from the diagonal line joining the source and target. Because of property of the algorithm, it is not able to compute paths when a complex structure of wall, as shown in Figure A.11, is present. Only 1 hopping point (that travels along the diagonal of the bounding rectangle) prevents this algorithm from computing path in presence of a complex structured wall. This is the reason why Blackmamba implementation enters infinite recursion for such obstacles and hence exceeds the 250ms timeout value during performance measurement tests.

Figure A.10: Transparent boundary of the game field
Figure A.11: Blackmamba fails to compute path for complex walls
B. Viper path finding algorithm

Proposed by: Suraj Sapkota

B.1 Definitions and Assumptions

This algorithm makes the following assumptions about the game field:

- The rectangle with the dotted border is the view port (vp) of each player. Other rectangles attached with it in each side are the virtual view port (vvp) in their corresponding sides.
- Small yellow box is a wall-unit. Group of attached wall-unit is said to be wall.
- The border of the game-field is transparent unless there is wall on the border.
- Wall is the only obstacle in the game field.
- There can be multiple wall in the same game-field.
- Finally considering all these, the aim of this algorithm is to find all (there may be multiple path of same distance) shortest path from the snake (head) to food. However as a result it will return only one among those paths.

Figure B.1: The game field (unfolded envelop)
B.2 Principle and implementation of this algorithm

This algorithm will not only deal with finding only one shortest path instead it calculates multiple path between the source and the destination, and all of these path have the same distance, the shortest distance. Actually the result is not in the form of multiple path, merely it is a collection of the points that it must pass through. It is the position of the points that make the path multiple. It is described later in more detail.

The algorithm proceeds as follows:

1. This algorithm begins with splitting all the game field into small rectangles called Fundamental open rectangle - FOR\(^1\), as shown in Figure B.2a and Figure B.2b, say \(R_1, R_2, R_3, ..., R_n\). If we represent the complete game-field (only view port), as \(G\) then mathematically the wall (\(W\)) can be defined as:
\[
W = G - R_1 \cup R_2 \cup R_3 \cup ... \cup R_n
\]

Broadly, this algorithm deals with splitting the game field into several (as many possible) rectangles such that, the union of all these small rectangles result in the game field that exclude wall. ie, the snake can move safely from any point in the rectangle(\(R_i\)) to any other point within the same rectangle(\(R_i\))and hence called Open Rectangles (OR).

2. Next we define Gate\(^2\), as in real life, as a way to move from a FOR to another adjacent FOR. Calculate the shortest distance between each two Gates of each FOR (eg: the shortest distance between the rectangles \(R_1\) and \(R_5\) are determined by the distance between the two gates of \(R_3\) shared with \(R_1\) and \(R_5\) respectively).

3. After the calculation of the shortest path, in above step, a graph as shown in Figure B.3 is formed. In the above graph:

   - The dotted lines defines direct connection, the direct path
   - The dark line shows a one step indirect connection between those rectangles that are not connected directly
   - \(S_{ij}\), beside with the vertex, (in graph) denotes that \(FOR_i\) is connected to \(FOR_j\) through this vertex and vice versa. This can also be called indirect path and it posses the path distance.
   - There may be multiple indirect path. For example: we can move form node 2 to node 5 (\(S_{25}\)) in two different ways: via node 3 or node 5

4. Finally after all these calculation, we can calculate the path between two FOR’s, the source and the destination with shortest distance. When two points (source and target) are given we can determine the FOR’s, they lie in and can compute the path between them.

A sample path computation depicted in Figure B.4, shows the path calculated by the algorithm from the snake (red one pointed by an arrow A) to the food (blue one pointed by an arrow F). The points pointed by the arrows A, B, C, D, E and F are points (Hoping Points) that the intelligent player must pass through. And thus allowing multiple possibility for path.

The path is multiple because, there exist multiple way to go from A to B and E to F. Further the rectangles as formed by the points A and B ( E and F ) can also be called Derived Open Rectangle (as it doesn’t contain any wall, and hence allows free movement of the snake inside it).

---

\(^{1}\) Open rectangle is a rectangle that does not contain any wall. Within it snake can move freely. The algorithm of generating FOR has been dealt with in following section.

\(^{2}\) A gate must be common to only two FOR
(a) Slitted into 6 small pieces of rectangle

(b) Broader view of game field slicing

Figure B.2: Slicing of game field into Fundamental Open Rectangle - FOR
Figure B.3: Graph with each node representing a Fundamental Open Rectangle - FOR and data associated with each node

Figure B.4: Path computation for a sample case
B.3 Algorithm to divide the game field into Fundamental Open Rectangle - FOR

The algorithm involves slicing the game-field in cyclic-order in anti-clockwise direction starting with the left edge. As the game field initially has no terminating edge (because of the envelope) we start with (visually) left-edge of the game field.

1. Initially starting from the zeroth coordinate move to both sides until it strikes to a wall. Name it Fundamental Open Rectangle (FOR\(_1\) and FOR\(_2\) or Just \(R_1\) and \(R_2\)) as shown in Figure B.5a and B.5a.

![Figure B.5: Slicing of game field into Fundamental Open Rectangle - FOR](image)

**Figure B.5: Slicing of game field into Fundamental Open Rectangle - FOR**

2. Now the game-field has shrunk to a form as shown in Figure B.6a. (It now has changed to cylindrical shape from envelope shape.)

3. Moving in anti-clockwise direction, slice from bottom in both direction (up and down) to get other two rectangle \(R_3\) and \(R_4\). Figure B.6b. Until this step the rectangle was not bounded (either in all four sides like an envelop or in two sides, top and bottom, like a cylinder). From this step onward the sliced game field will be bounded as shown in Figure B.7a.

![Figure B.6: Slice from bottom in both direction (up and down in anti-clockwise direction)](image)

**Figure B.6: Slice from bottom in both direction (up and down in anti-clockwise direction)**

4. Now try to slice rectangle from top left, bottom-left, bottom right and top-right respectively. Here we fail to slice the rectangle from top-left and bottom-right as there is wall(-unit) at
the edge. We can now create two new rectangles from bottom-left and top-right namely R5 and R6 as shown in Figure B.7b

5. Now slice away the boundary layer of the game-field formed in above step as shown in as shown in Figure B.7c

6. As earlier again try to slice the game-field in the anticlockwise direction starting from left. It will result in two more rectangle R7 and R8 as shown in Figure B.7d.

During this we assume already sliced rectangle as a hollow space.

(a) the sliced game field is bounded now
(b) STEP 4: FOR R5 and R6 marked
(c) slice away the boundary layer of the game-field and R8 marked
(d) STEP 2: FOR R7 marked

Figure B.7: Slicing of bounded game field

7. For Further Optimization we proceed by breaking down the rectangle that touches itself (eg: in this case, the top edged of rectangle R1 touch itself to its bottom edge) as shown in Figure B.8. These type of rectangles are divided in the middle. This results in best utilization of the transparency of the Game Field.

Figure B.8: Optimization of the result of slicing procedure

B.3.1 Special Case: Game Field with no wall

If there is no wall we simply chop the game field vertically and horizontally from the middle resulting in 4 FOR’s. This results in maximum utilization of the transparency of the game field. The result is shown in Figure B.9.
B.4 **Salient Features**

- As most of the processing is done before the actual game starts, it must reduce the the processing time during the game-time.

- The concept of open-rectangle allows multiple path, and hence within it the snake can be moved randomly towards the specified point. This provides the snake with some degree of intelligence.

- Furthermore, during the calculation of the path the snake can move within the FOR in which it currently lie.

B.5 **Limitation**

The process of path computation becomes complex and slow when the number of Fundamental Open Rectangle - FOR, produced during game field slicing, is very high.
C. Inter Snake Communication Protocol (ISCP)

The communication between the game server and client applications is done using the Inter Snake Communication Protocol (ISCP). The protocol contains five different classes. These are listed below:

**Information** Used for transmission of player’s information (name, color, location). Used only when the player creates/joins the game.

**ChatMessage** Transmits the chat message. Works only before the actual game starts.

**ControlSignal** Transmits control signals between server and clients. These signals are mainly used for synchronization.

**LevelData** Sends data required on level change (food, wall, score, life, etc). Sent initially when the actual game begins and then after every level change.

**Move** Transmits the movement of clients, collision information and also the information about food capture.

The communication packets are encoded in byte format for the purpose of transmission and are decoded in similar fashion. The encoding and decoding of packets in client side is handled by ClientEncoder and ClientDecoder respectively. Similar for game server, encoding and decoding is handled by ServerEncoder and ServerDecoder respectively.

For instance, the server encoder for chat message is known as ChatServerEncoder. The encoders and decoders are placed under net.sf.isnake.codec in the source tree.

C.1 Types of communication packet in ISCP

On the basis of communication packet flow, we can classify the communication basically into two types:

- **Client Initiated Communication**: In this type of communication, the client first encodes the packet with necessary data. Send it to server in flipped order. Flipping is required so that it gets received in correct order at the other end. The server decodes it. The server re-encodes the data with some further information (if required such as sender’s Id) and flips it. This packet is re-transmitted to clients where it is decoded. The communication cycle is shown in Figure C.1a.

- **Server Initiated Communication**: In this type of communication, server first encodes the packet with necessary data and transmits in flipped order. The packet is decoded on client side. The communication cycle is shown in Figure C.1b.

The five classes of communication can send various messages. All the messages are decoded the same way they are encoded.
C.2 Conventions used in the protocol definition

Following conventions are used in depicting the encoded packet in this paper:

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&gt;</td>
<td>Shows fields to be sent along with identifier</td>
</tr>
<tr>
<td>id</td>
<td>Represents the player id</td>
</tr>
<tr>
<td>[]</td>
<td>The fields within are part of an array in the protocol class</td>
</tr>
<tr>
<td>()</td>
<td>Shows the data type of the field</td>
</tr>
</tbody>
</table>

C.3 Protocol Definition

C.3.1 Class: Chat Message

1. Packet : Chat Text

   Identifier : CH  
   Type : Client Initiated  
   Description  
   Is used to transmit a chat message typed by a client to other clients. to_id is required for one to one communication which is not currently allowed in the game but has been designed for future enhancements. 0 is being sent as to_id which is basically a multi cast id.

   ClientEncoder  
   CH<to_id(byte)><len_of_message(short)><message(string)>  

   ServerEncoder  
   CH<from_id(byte)><to_id(byte)><len_of_message(short)><message(string)>

C.3.2 Class: Information

1. Packet : Player info

   Identifier : IN
Type : Client Initiated
Description
Is used to transport the information of a player viz; name, color and location.

ClientEncoder
IN<info_len(short)><name:color:location(string)>
ServerEncoder
IN<info_len(short)><id:name:color:location(string)>

C.3.3 Class: ControlSignal

1. Packet : Acknowledgment

Identifier : AK
Type : Client Initiated
Description
This packet is sent when a client successfully receives Level data. The status_code field has been reserved for future enhancements. As of now it is sent as 0.

ClientEncoder
AK
ServerEncoder
AK<id(byte)><status_code(byte)>

2. Packet : Quit

Identifier : QT
Type : Client Initiated
Description
Sent when a client quits the game.

ClientEncoder
QT
ServerEncoder
QT<id(byte)>

3. Packet : Ready

Identifier : RY
Type : Client Initiated
Description
Is sent when the user specifies his readiness to start the game.

ClientEncoder
RY
ServerEncoder
RY<id(byte)><status_code(byte)>

51
4. Packet : ID

Identifier : ID
Type : Server Initiated
Description
This packet is sent by server to client to assign an id to the player. This is done as soon as the client sends his information.

ServerEncoder
ID<id(byte)>

5. Packet : Start

Identifier : ST
Type : Server Initiated
Description
Represents start of various events in the game on the basis of status_code. status_code 0 represents the beginning of transmission of level data. status_code 1 represents the beginning of level synchronization task before beginning of a level and status_code 2 is a signal to start the actual level play.

ServerEncoder
ST<status_code(byte)>

6. Packet : Stop

Identifier : SP
Type : Server Initiated
Description
Represents the end of level.

ServerEncoder
SP<status_code(byte)>

7. Packet : Force

Identifier : FR
Type : Server Initiated
Description
This communication packet forces a player to send a ready packet when all others have expressed their willingness to begin the game.

ServerEncoder
FR<id(byte)><status_code(byte)>

8. Packet : No Acknowledgment
**Identifier** : NK
**Type** : Server Initiated
**Description**
This packet is sent by server when the information supplied by client is not valid. status_code here represents field which has invalid data. Invalid data occurs when information field contains reserved characters or duplicate entry.

ServerEncoder
NK<id(byte)><status_code(byte)>

**C.3.4 Class: LevelData**

1. Packet : Begin from

**Identifier** : BF
**Type** : Server Initiated
**Description**
The packet sends starting coordinate of all the players in a single packet. The player begins level from this coordinate and is reset to this coordinate on collision.

ServerEncoder
BF<3*no_of_players(byte)>[<id(byte)><x_coordinate(byte)>
<y_coordinate(byte)>]

2. Packet : Score

**Identifier** : SC
**Type** : Server Initiated
**Description**
Sends score of all the players in the packet on beginning of a level.

ServerEncoder
SC<8*2*no_of_players(byte)>[<id(long)><score(long)>]

3. Packet : Level

**Identifier** : LV
**Type** : Server Initiated
**Description**
Sends the level number to the clients.

ServerEncoder
LV<level_number(byte)>

4. Packet : Life
Identifier : LF
Type : Server Initiated
Description
Sends the life count for each player on beginning of a level.

ServerEncoder
LF<2*no_of_players(byte)><id(byte)<life_count(byte)>]

5. Packet : Wall

Identifier : WL
Type : Server Initiated
Description
Sends the wall coordinates to the clients.

ServerEncoder
WL<2*wall_length(short)><x_coordinate(byte)<y_coordinate(byte)>]

6. Packet : Food

Identifier : FD
Type : Server Initiated
Description
Sends the food coordinates to the clients on the beginning of a level.

ServerEncoder
FD<x_coordinate(byte)<y_coordinate(byte)>]

C.3.5 Class: Move

1. Packet : Collide

Identifier : CL
Type : Client Initiated
Description
The packet is sent when a player looses a life. Possibility of more than one players colliding on same turn has been taken into account. At present, however, we are checking the collision on server and the communication is thus Server Initiated.

ClientEncoder
CL
ServerEncoder
CL<no_of_players_collided(byte)>[<collided_player_id(byte)>]

2. Packet : Move
Identifier : MV
Type : Client Initiated
Description
Carries the change in x and y (dxdy) of the clients to the server.
Server merges change in coordinate of all the clients in a new MV packet and transmits to the clients.

ClientEncoder
MV[id(byte)<dx(byte)<dy(byte)>]

ServerEncoder
MV[3*no_of_players(byte)>[<id(byte)<dx(byte)<dy(byte)>]]

3. Packet : Eatens

Identifier : ET
Type : Server Initiated
Description
Server sends the packet whenever the food gets eaten. This packet also initiates the generation of Food packet as a level can have more than one food.

ClientEncoder
ET

ServerEncoder
ET[eaten_id(byte)]
FD[x_coordinate(byte)<y_coordinate(byte)>]