RECOGNISING ACHIEVEMENT

Exemplar Candidate Work

GCE Computing

OCR Advanced GCE in Computing H447

Unit: F454 Computing Project

© OCR 2011

Contents

Contents 2

Introduction 3

Moderator's Commentary: 4

Candidate's work 6

Introduction

This exemplar material serves as a general guide. It provides the following benefits to a teacher:

- Gives teachers an appreciation of the variety of work that can be produced for this unit
- Shows how the mark scheme has been applied by a senior assessor
- Provides examples of both good and weak application of different parts of the mark scheme
- Provides real examples of work conducted under controlled assessment conditions.

It is important to make the point that the teacher support materials play a secondary role to the Specification itself. The Specification is the document on which assessment is based and specifies what content and skills need to be covered in delivering the course. At all times, therefore, this teacher support should be read in conjunction with the Specification. If clarification on a particular point is sought then that clarification should be found in the Specification itself.

Moderator's Commentary:

Section a

An interesting start, the result and the approach are both declared in the first sentence, this would have suggested problems under the previous specification and that there was unlikely to be any realistic end user involvement in the design and development process. In this case, as we read on, the project clearly identifies a suitable end user group. The background to the proposed project is clearly set out following on from some very interesting research into the development of the game so far. The section then clearly includes excellent interaction with the end user and a full analysis of their feedback leading to a detailed requirements specification including the hardware and software.

The evidence may not be in clearly delineated sections but it is presented clearly and logically. There is evidence of good research beyond a basic interview and we are left in no doubt about the validity of the end user involvement, and what the project is all about.

Mark: 13 (Max 14)

Section b

There is a fully detailed design including measureable objectives, evidence of end user involvement and of an iterative process leading to a fully agreed final design. Any equally skilled programmer would be able to take this design and produce the same (or at least very similar) end product. This is an excellent example of the design process .

There is a test strategy explain how the system will be tested as it is developed including a detailed test plan. I would have liked to see how the test plan was designed to be used during the development phase to check each element of the process and what data would be used to complete the final testing. This is a minor point but one worth making.

There is a complete set of algorithms and we are left in no doubt that this has been thoroughly checked to establish that it is a solution to the problem.

Mark: 16 (Max 16)

Section c (i)

There is mention of some testing and modification during development though there could be more evidence here of the development process, evidence in the later sections demonstrates the development process adequately and should be credited here.

The test plan is applied to the finished product and there is copious evidence to show that this has been done and that the system works. For development what we have is some mention of a break point in the code, annotated code and a brief discussion of the process. This might have been better presented as an iterative process showing the stages in the development of the system. There are some minor issues with the evidence presented hence:

Mark: 15 (Max 16)

Section c (ii)

It is clear that the system has been fully tested post development, that there has been some end user testing, though little evidence of the end user testing is available (perhaps this is limited by the use of paper based evidence). The response to the testing is excellent and there is evidence of improvements / modifications being made to the system in light of the test results. An excellent section.

Mark: 14 (Max 14)

Section d

There is excellent on screen help for such a system and all the additional documentation that could be required is available, hence

Mark: 10 (Max 10)

Section e

The evaluation discusses the system objectives one by one with evidence of what was achieved.

The response to the end user testing shows a summary of the end user feedback and there is some brief response to the largely positive feedback and to the minor criticisms

The good and bad points are clearly identified and there is some discussion about how the negatives can be resolved.

Mark: 10 (Max 10)

Total mark for portfolio: 78 (Max 80)

Candidate’s work

A2 Computing Project

Contents

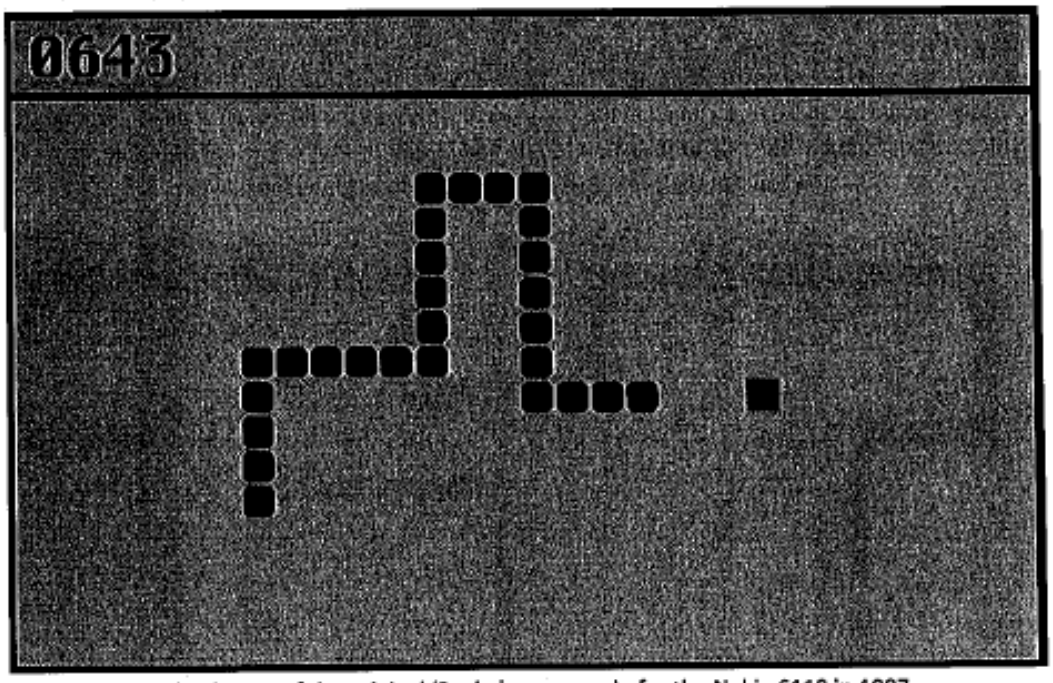
Section 1- Definition, investigation and analysis....3
Problem definition....3
Current systems process model....5
Selection of user group....6
Investigation and analysis....7
Initial interview results....7
Initial interview analysis....8
Second interview....9
Results of second interview....11
Analysis of second interview results....14
Requirements specification....14
Hardware and software requirements....15
Section 2- Design -Nature of the solution....16
Design objectives....16
Interface Design 1: Level Selection screen....18
Level selection screen interview....20
Interface Design 2: Instructions screen....21
Instructions screen interview....24
Interface Design 2: Gameplay screen....25
Gameplay screen interview....27
Amendments to design objectives....27
Modular design of snake – structure diagram....30
Data structure design....31
Test strategy & test plan....33
Algorithms & testing....37
Section 3- Software development and testing....47
Part 1 – Software development....47
Part 2 – Testing....51
Screenshots from testing....58
Result of testing....84
Evidence of improvements....86
User testing....87
Annotated code – form named frmSnake....90
Annotated code – form named frmInstructions....98
Annotated code – module named mdlSnake....99
Section 4- Additional documentation....Inserted after page 99
Section 5- Evaluation....100
Part 1 – The degree of success in meeting the original objectives....100
Part 2 – An evaluation of the user's response to the system....103
Part 3 - Desirable extensions....104

Section 1 – Definition, investigation and analysis

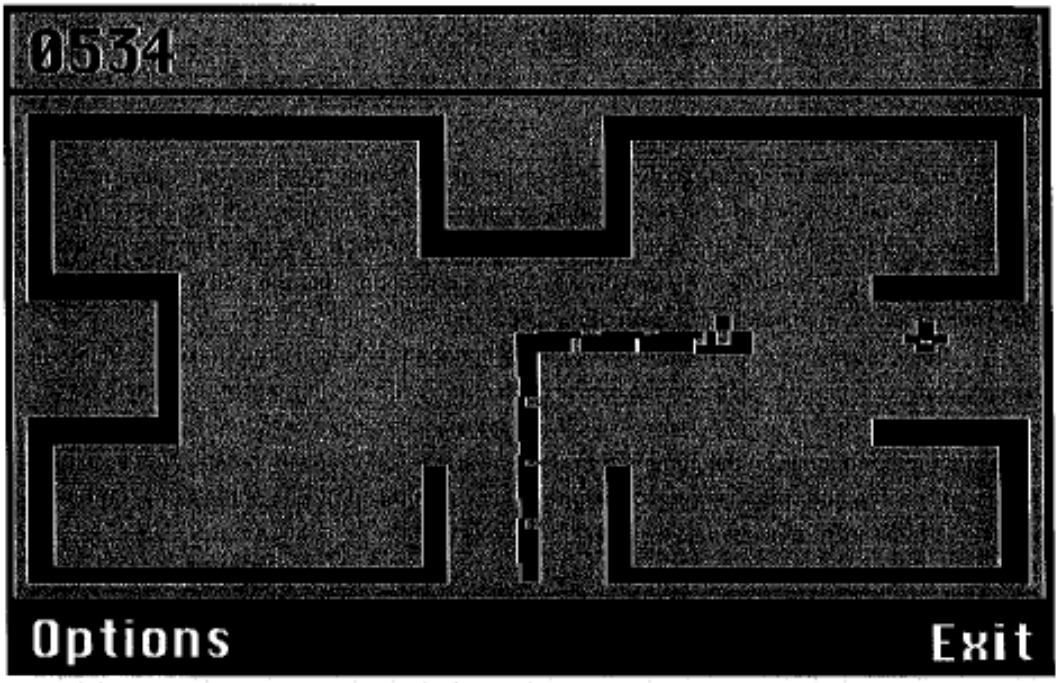
Problem Definition

Using Visual Basic 6.0, I am going to develop a version of the computer game commonly known as "Snake" that was popularised by its inclusion in Nokia mobile phones from 1997 onwards. The game allows the user to control the movement of a 'snake' (represented by a series of connected blocks called 'bits') that is constantly in motion, by adjusting the direction that the snake's head is facing. The snake's 'body' then follows the path of the head.

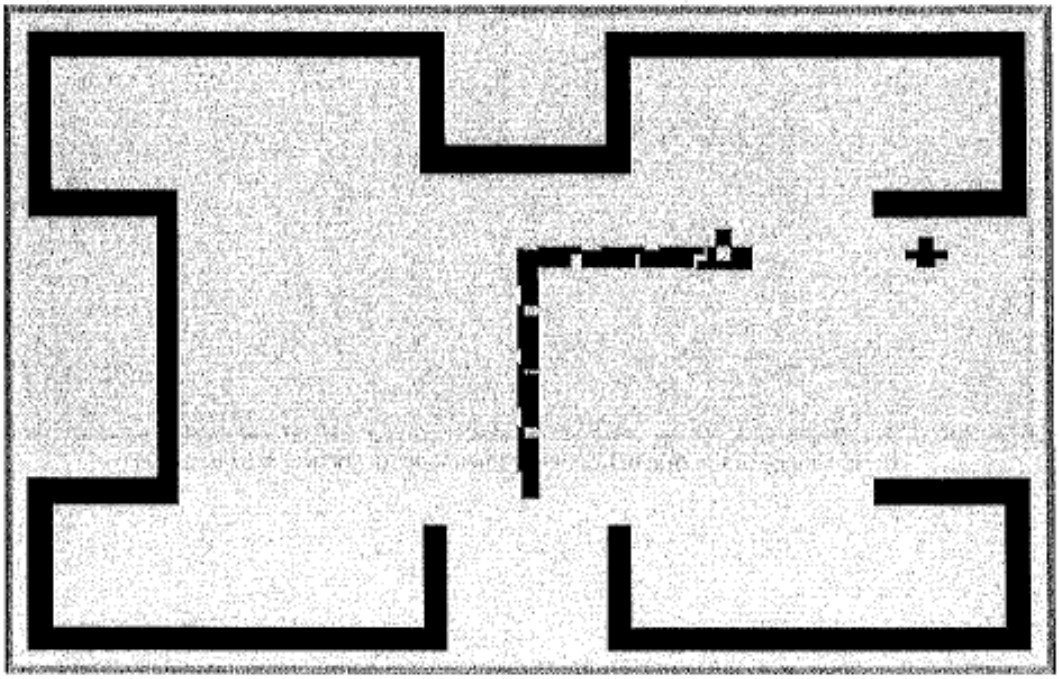
In most versions of Snake, the game ends when the user crashes the snake's head into its trailing body, or when the snake's head comes into contact with the borders of the game screen. The objective of the game is to collect 'food' by directing the snake's head onto food icons that appear (one at a time) randomly on the game screen, thus awarding the user with points. With each piece of food that is eaten, the snake becomes longer thus increasing the difficulty of the game. Here I shall include screen shots from various versions of the Snake game.



Design image of the original 'Snake' game made for the Nokia 6110 in 1997



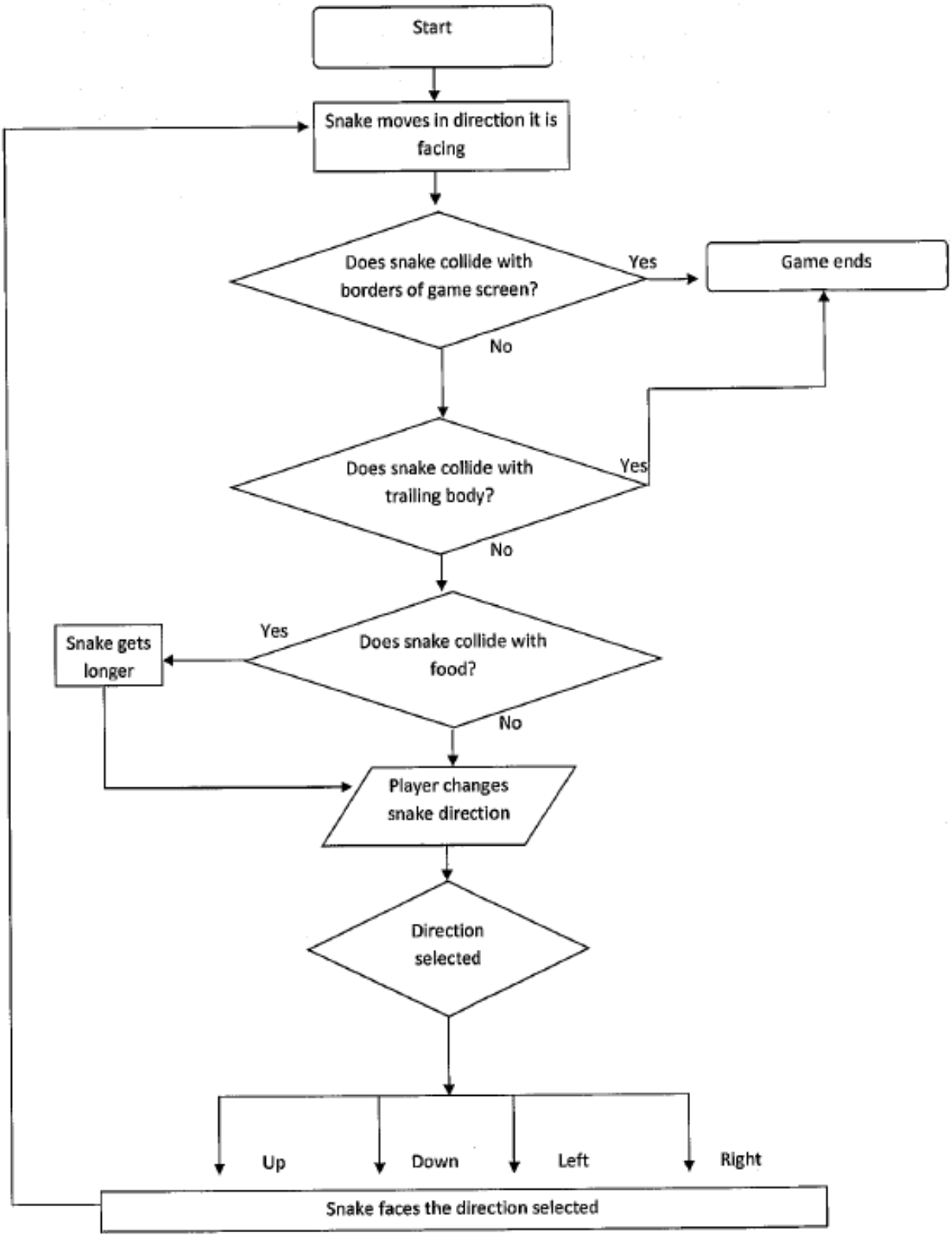
Design image of 'Snake II', included in the Nokia 3310. This was the first snake game to include the ability to go 'through' one side of the game screen and re-emerge out of the opposite side, as well as the concept of 'levels' that are advanced through by the player.



Design screen from 'Snake EX', the first snake game to include colour.

Current systems

I shall now illustrate using a simple process model the gameplay mechanic of a typical Snake game.



Selection of user group

My end user will be anyone who wishes to play the game – most probably students. As such, I shall correspond with students from sixth form colleges with regards to user requirements and feedback throughout development. There are many features that can be included a Snake game, for example the ability to go 'through' one side of the game screen and re-emerge out of the opposite side, or to have different difficulty levels. In order to discover what features are desired by my user group and to thereby develop a requirements specification I shall conduct a series of interviews on students aged 17-18.

Investigation and analysis

I intend to interview a range of students aged between 17 and 18, from various sixth form colleges and these are the questions that I intend to ask:

- Have you ever played a version of 'Snake'?
- If so, how many different versions of the game have you played?
- What features (for any of the versions that you have played) did you like?
- What features, if any, did you dislike?
- Are there any novel features that you would like to see in a new version of 'Snake'?

Initial interview results

I have interviewed seven students and here I shall compile all sensible responses, in order of the frequency with which they occurred. The frequency of each response is written in parentheses. I shall treat this group of students as my user group and will therefore return to them for feedback throughout development. Their names are: Thomas Bates, Sebastian Zepata, Sunoo Park, Kristian Dainton, Joseph Rogers, Abhiram Nandakumar and Louise Lord

Have you ever played a version of 'Snake'?

- Yes (7)

If so, how many different versions of the game have you played?

- At least three (4)
- Two (2)
- One (1)

What features (for any of the versions that you have played) did you like?

- The ability to change the speed of the snake (5)
- A large screen size (5)
- The ability to advance through 'levels' of the game, each with varying difficulty (3)
- The inclusion of different kinds of 'food' for the snake that awarded the player with additional points (2)
- A multiplayer mode (1)
- The ability to go through walls (1)

What features, if any, did you dislike?

- The ability to go through walls – because it made the game too easy (5)
- The lack of colour (4)
- Brevity of the game (1)

Are there any novel features that you would like to see in a new version of 'Snake'?

- The ability to advance through 'levels' of the game (3)
- Improved aesthetics – e.g. inclusion of colour (3)
- Ability to 'save' the game in order to return to a particular snake speed and length (3)
- Ability to change starting length of the snake (2)
- Ability to change speed of snake (2)
- Inclusion of food items that penalised the player, e.g. by making the screen turn black (1)

Analysis of initial interview results

All seven interviewees had played 'Snake' in at least one of its many forms, and many of them had played several different versions. This means that although there are multiple features that the interviewees liked/disliked, they were not all necessarily referring to the same version of 'Snake'.

For this reason, it would not be sensible to include all features that were liked, as they may not all be compatible for simultaneous inclusion in one version of 'Snake'. Also, not all requested features can be included as they may be incompatible with other features. Instead, I shall create a new version of 'Snake' that contains a combination of the most popular features, together with the most commonly requested new features (that would be compatible with the other features). By 'compatible' I mean, for example, that having a large screen size would be incompatible with the ability to go through walls, as this would make the game too easy; whereas having a large screen size is perfectly compatible with the ability to change the speed of the snake.

Some interviewees had conflicting opinions, for example the ability to go through walls was a feature that was liked by some and disliked by others. In considerations such as these, I shall go with the preference of the majority (provided their preference is compatible with other features to be included.)

Based on the results of the initial interview, together with considerations of feature compatibility, I have decided upon the following requirements:

Basic Requirements

1) The game ends when the snakes head collides with its body
2) The snake is constantly in motion
3) The player is able to control the direction that the head is facing
4) The snake's body follows the path of the head
5) The player is able to direct the head of the snake onto 'food' items
6) Each time a food item is eaten, another will be placed randomly on the game screen

Variable features

7) The screen size is large
8) The game ends when the snake's head collides with the borders of the game screen
9) The player progresses through 'levels' of the game. The level number is incremented each time a certain number of food items are 'eaten'
10) At each new level the speed and length of the snake is increased
11) The player is able to choose the level at which he/her begins the game from (therefore the player will be able to control the length and speed of the snake, and this also allows a user to continue a previous game in the sense that they will be able to start from a speed and length that they were previously on)
12) Colours (that have been determined by the user group) will be used

Second interview

To determine whether the variable features satisfy my user group's requirements, and to decide upon details such as the number of food items to be eaten in order for a level to be incremented, and to obtain values to be used for the size of each snake 'bit' etc, I shall conduct another interview on each of the seven students of my user group.

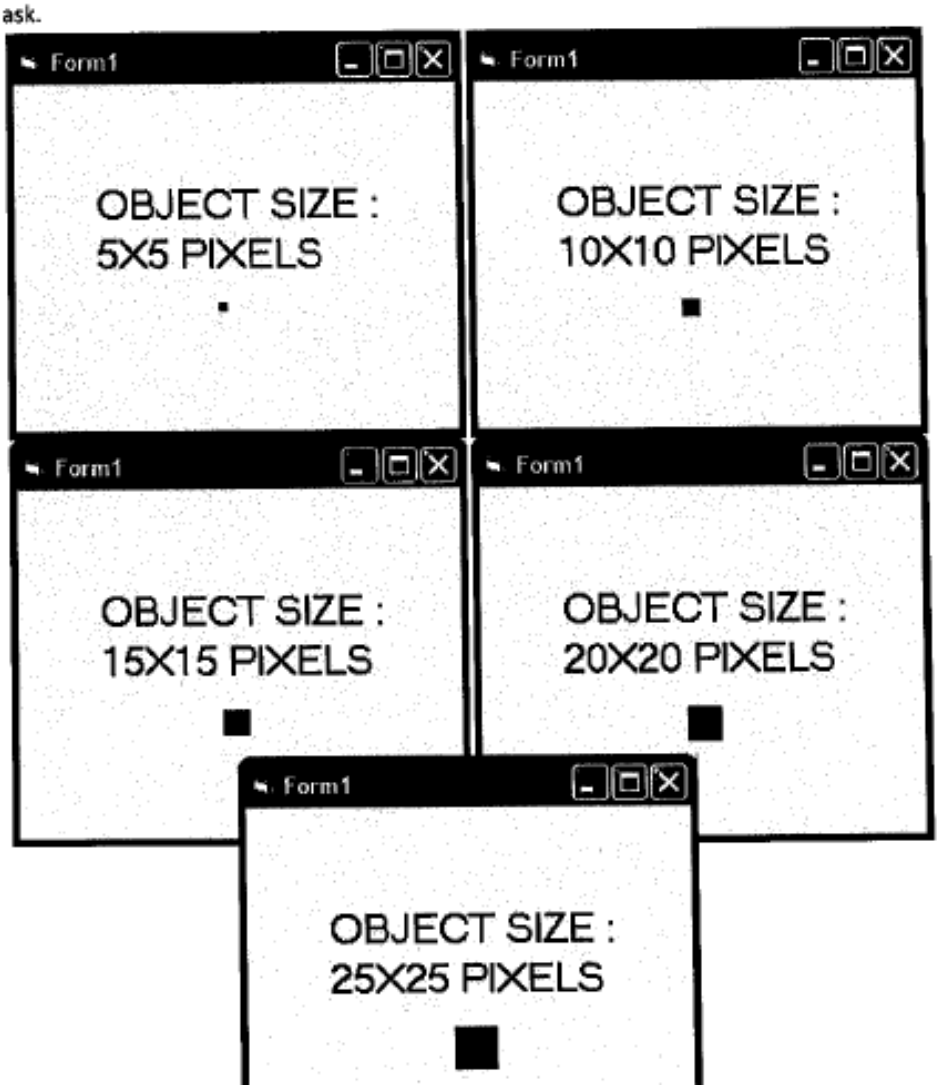
These are the questions that I intend to ask:

- Are you satisfied with the variable features listed?
- If not, what changes would you wish to be made?
- How large would you like the screen size for the game to be (relative to monitor size of resolution 1280x1024)?
    - Approximately 1/2 size (600x600)
    - Approximately 1/3 size (400x400)
    - Approximately 3/4 size (900x900)
    - Other
- If the player is to advance through levels of increasing snake length and speed, how many 'food' items should be eaten by the snake before the level is incremented?
- Approximately how large should each snake 'bit' be?
    - Very small (5x5 pixels)
    - Small (10x10 pixels)
    - Medium (15x15 pixels)
    - Large (20x20 pixels)
    - Very large (25x25 pixels)
    - Other
- How far (in pixels) should the snake's minimum movement be?
- Approximately how large should each food item be?
    - Very small (5x5 pixels)
    - Small (10x10 pixels)
    - Medium (15x15 pixels)
    - Large (20x20 pixels)
    - Very large (25x25 pixels)
    - Other
- By how many 'bits' of length should the snake increase at each level?
- How many 'bits' should the snake consist of at the beginning of level 1?
- With regards to the increase of snake speed at each level, how great do you think the change should be - i.e. would you rather the game have a shallow, medium or steep difficulty curve? (Shallow = decrease in time interval between snake movements of 10 milliseconds, Medium = decrease in time interval between snake movements of 20 milliseconds, Steep = decrease in time interval between snake movements of 30 milliseconds)
- What should the time interval between snake movements be for level 1?
    - 100

- 200
- 300

- How many levels should you be able to choose from?
- How many levels should there be in total?
- Should the player use a keyboard or a mouse to control the snake?
  - If Keyboard: Which keys should be used to control the snake?
  - If Mouse: Should the player control the snake by moving the mouse or by clicking on areas of the screen?
- Should the player be able to lose the game by making the snake go back on itself (i.e. by changing the direction that the head is facing from right to left)?
- What colour(s) would you like to see used for the snake, the food items and the background?
- What should be displayed on the initial level selection screen?
- How should the level be selected?
- Which direction should the snake's head initially face?

To demonstrate aesthetic features of size that may seem abstract to my users, I shall show them the following screenshots of a Visual Basic form with objects of sizes corresponding to the questions I ask.



Results of second interview

Again, here I shall compile all responses. The frequency of each response is written in parentheses.

Are you satisfied with the variable features listed?

- Yes (6)
- No (1)

If not, what changes would you wish to be made?

- The game should not end when the snake collides with the borders of the game screen, but the player should be able to 'go through' these borders and re-emerge out of the opposite side of the game screen (1)

How large would you like the screen size for the game to be (relative to monitor size)?

- Approximately 3/4 size (900x900) (3)
- Approximately 1/2 size (600x600) (2)
- Approximately 1/3 size (400x400) (2)
- Other (0)

If the player is to advance through levels of increasing snake length and speed, how many 'food' items should be eaten by the snake before the level is incremented?

- Five (4)
- Four (2)
- Seven (1)

Approximately how large should each snake 'bit' be?

- Very small (5x5 pixels) (1)
- Small (10x10 pixels) (5)
- Medium (15x15 pixels) (1)
- Large (20x20 pixels) (0)
- Very large (25x25 pixels) (0)
- Other (0)

How far (in pixels) should the snake's minimum movement be?

- Same as length of snake bit (7)
- Other (0)

Approximately how large should each food item be?

- Very small (5x5 pixels) (1)

- Small (10x10 pixels) (5)
- Medium (15x15 pixels) (0)
- Large (20x20 pixels) (1)
- Very large (25x25 pixels) (0)
- Other (0)

By how many 'bits' of length should the snake increase at each level?

- Six (3)
- Two (2)
- Five (2)

How many 'bits' should the snake consist of at the beginning of level 1?

- Thirty (4)
- Twenty (2)
- Ten (1)

With regards to the increase of snake speed at each level, how great do you think the change should be - i.e. would you rather the game have a shallow, medium or steep difficulty curve? (Shallow = decrease in time interval between snake movements of 10 milliseconds, Medium = decrease in time interval between snake movements of 20 milliseconds, Steep = decrease in time interval between snake movements of 30 milliseconds)

- Shallow (5)
- Medium (2)

What should the time interval between snake movements be for level 1 (in milliseconds)?

- 100 (5)
- 200 (2)
- 300 (0)

How many levels should you be able to choose from?

- Thirty (1)
- Unlimited (2)
- Ten (4)

How many levels should there be in total?

- Unlimited (6)
- Fifty (1)

Should the player use a keyboard or a mouse to control the snake?

- Keyboard (7)
  - Keyboard: Which keys should be used to control the snake?
    - Directional buttons (7)

Should the player be able to lose the game by making the snake go back on itself (i.e. by changing the direction that the head is facing from right to left)?

- No (7)

What colour(s) would you like to see used for the snake, the food items and the background?

Snake

- Green (4)
- Red (1)
- Purple (1)
- Don't mind (1)

Food

- Pink (4)
- Red (2)
- Yellow (1)

Background

- Black (4)
- Blue (2)
- Green (1)

What should be displayed on the initial level selection screen?

- Instructions on how to play the game (3)
- Instructions on how to select a level (3)
- Don't mind (1)

How should the level be selected?

- With a drop down list and combo box (4)
- With a text box and combo box (2)
- Don't mind (1)

Which direction should the snake's head initially face?

- Right (7)

Analysis of second interview results

The vast majority of my user group was satisfied by the variable features that I had decided upon, and for this reason I shall ignore the single objection.
With regards to the preferred values for the variable features, I shall go with the preference of the majority in order to create a requirements specification.

Requirements Specification

User Requirements

1) Design requirements
    a. The screen size is large – (900x900 pixels)
    b. The colour of the snake will be: green
    c. The colour of the food will be: pink
    d. The colour of the background will be: black
    e. Each 'bit' of the snake will be 10x10 pixels in size
    f. The snake should start at a length of thirty 'bits' on level 1
    g. The snake's head should initially be facing right
    h. Each food item will be 10x10 pixels in size
    i. The time interval between each snake movement at level 1 will be 100 milliseconds
    j. There is no pre-defined limit on the number of levels to be played through (instead this is limited by the skill of the player)
2) Input requirements
    a. The player is able to choose the level at which he/she begins the game from (between 1 and 10) with the use of a drop down list and combo box
    b. The player is able to control the direction that the head is facing with the use of the directional keys on a keyboard, however the snake cannot go back on itself
3) Processing requirements
    a. The coordinates of the snake's head need to be continually incremented by 10 pixels in the direction that the snake's head is facing at a time interval that is determined by the level chosen
    b. Each snake bit behind the head must be incremented by 10 pixels in the direction that the snake's head was facing on a previous move (with the move number corresponding to the snake bit's distance from the head, such that the snake's body will follow the path of the head)
    c. It must be determined with each movement whether the snake's head collides into its body, in which case the game should end
    d. It must be determined with each movement whether the snake's head collides into the borders of the game screen, in which case the game should end
    e. It must be determined with each movement whether the snake's head collides into a food item, in which case the a the food item will be randomly moved to another position on the game screen

f. The system must record the amount of food items eaten, and increment the level number every time 5 food items are eaten
g. The time intervals between each movement of the snake must be decreased by 10 milliseconds every time the level number is increased
h. At each new level the length of the snake must be increased by 6 bits

4) Output requirements
   a. The current level number will be displayed
   b. The positions of each snake bit will be displayed in real time
   c. The position of the food item will be displayed
   d. Instructions on how to select a level and play the game will be displayed on the initial level selection screen

Hardware and Software Requirements

The system will be written using Visual Basic 6.0, and so as a guide the hardware and software requirements of the system should be similar to those necessary to run Visual Basic 6.0, together with hard disk space to store the game itself. I have obtained the minimum requirements needed to run Visual Basic 6.0 from the Microsoft website. The additional hardware requirements of the game will be: A keyboard to be used to control the direction that the snake's head is facing, a mouse to select the level and additional hard disk space to store the snake game itself.

Software Requirements	Justification
Microsoft Windows 95 or later operating system	An operating system required to run Visual Basic 6
Visual Basic 6 DLLs	Software required to run the Snake game

Hardware Requirements	Justification
PC with a 486DX/66-MHz or higher processor	Minimum processing power needed to run Visual Basic 6
16 MB of RAM for Windows 95 (32 MB recommended)	Minimum RAM needed to run Visual Basic 6
10 MB of hard disk space	Should be sufficient to store snake game
VGA or higher-resolution monitor; Super VGA recommended	Needed to display output from the game
Microsoft Mouse or compatible pointing device	Needed for user to select level
Computer Keyboard	Needed for user to give an input in order to change the direction that the snake's head is facing

User review of requirements

All seven of my users have reviewed and agreed to the above requirements.

Signed: T. Pares Seb Zapata Krishan Dainton
Louise Lord

Section 2 -Design -Nature of the solution

Design objectives

The design objectives are similar to the requirements previously specified. However, as a design consideration I have decided that the best way to display instructions on how to play the game would be in a new form that is made visible when a command button on the initial level selection screen is clicked, so that the initial screen is not cluttered with text.

To demonstrate this to my users and get their feedback on interface specifics such as these, I shall design the interfaces, display them in turn to my users and conduct a series of further interviews on each of my seven users with regards to the interfaces.

In addition, if the time interval between each snake movement is to be 100 milliseconds at level 1 and the time interval is to be decreased by 10 milliseconds every time the level number is increased then the time interval cannot decrease in this way beyond level 10. Therefore, as a processing consideration I have determined that the time interval should not decrease beyond level 10. I have discussed and agreed this consideration with all 7 members of my user group; their signatures are below the objectives.

1) Aesthetic considerations
    a. The screen size will be large – (900x900 pixels)
    b. The colour of the snake will be: green
    c. The colour of the food will be: pink
    d. The colour of the background will be: black
    e. Each 'bit' of the snake will be 10x10 pixels in size
    f. The snake should start at a length of thirty 'bits' on level 1
    g. The snake's head should initially be facing right
    h. Each food item will be 10x10 pixels in size
    i. The time interval between each snake movement at level 1 will be 100 milliseconds
    j. There is no pre-defined limit on the number of levels to be played through (instead this is limited by the skill of the player)
2) Input considerations
    a. The player will be able to choose the level at which he/she begins the game from (between 1 and 10) with the use of a drop down list and combo box
    b. The player will be able to control the direction that the head is facing with the use of the directional keys on a keyboard
    c. The player will not be able to make the snake go back on itself by pressing left when the head is facing right, or up when the head is facing down etc.
    d. The player will be able to view an instructions screen that opens in a new form by clicking a command button that is on the initial menu screen
    e. The player will be able to go back to the initial level selection screen from the instructions screen with the use of a command button

3) Processing considerations
    a. The coordinates of the snake's head will be continually incremented by 10 pixels in the direction that the snake's head is facing at a time interval that is determined by the level chosen
    b. Each snake bit behind the head will be incremented by 10 pixels in the direction that the snake's head was facing on a previous move (with the move number corresponding to the snake bit's distance from the head, such that the snake's body will follow the path of the head)
    c. It must be determined with each movement whether the snake's head collides into its body, in which case an error message should be produced and the game should end
    d. It must be determined with each movement whether the snake's head collides into the borders of the game screen, in which case an error message should be produced and the game should end
    e. It must be determined with each movement whether the snake's head collides into a food item, in which case the a the food item will be randomly moved to another position on the game screen
    f. The system must record the amount of food items eaten, and increment the level number every time 5 food items are eaten
    g. Up to and including the transition to level 10, the time intervals between each movement of the snake must be decreased by 10 milliseconds every time the level number is increased. This will be done using a timer to control the snake movements.
    h. At each new level the length of the snake must be increased by 6 bits
4) Output considerations
    a. The current level number will be displayed
    b. The positions of each snake bit will be displayed in real time
    c. The position of the food item will be displayed
    d. Instructions on how to select a level be displayed on the initial level selection screen
    e. Instructions on how to play the game will be displayed in a new form when the instructions command button is clicked

User review of objectives

All seven of my users have reviewed and agreed to the above objectives.

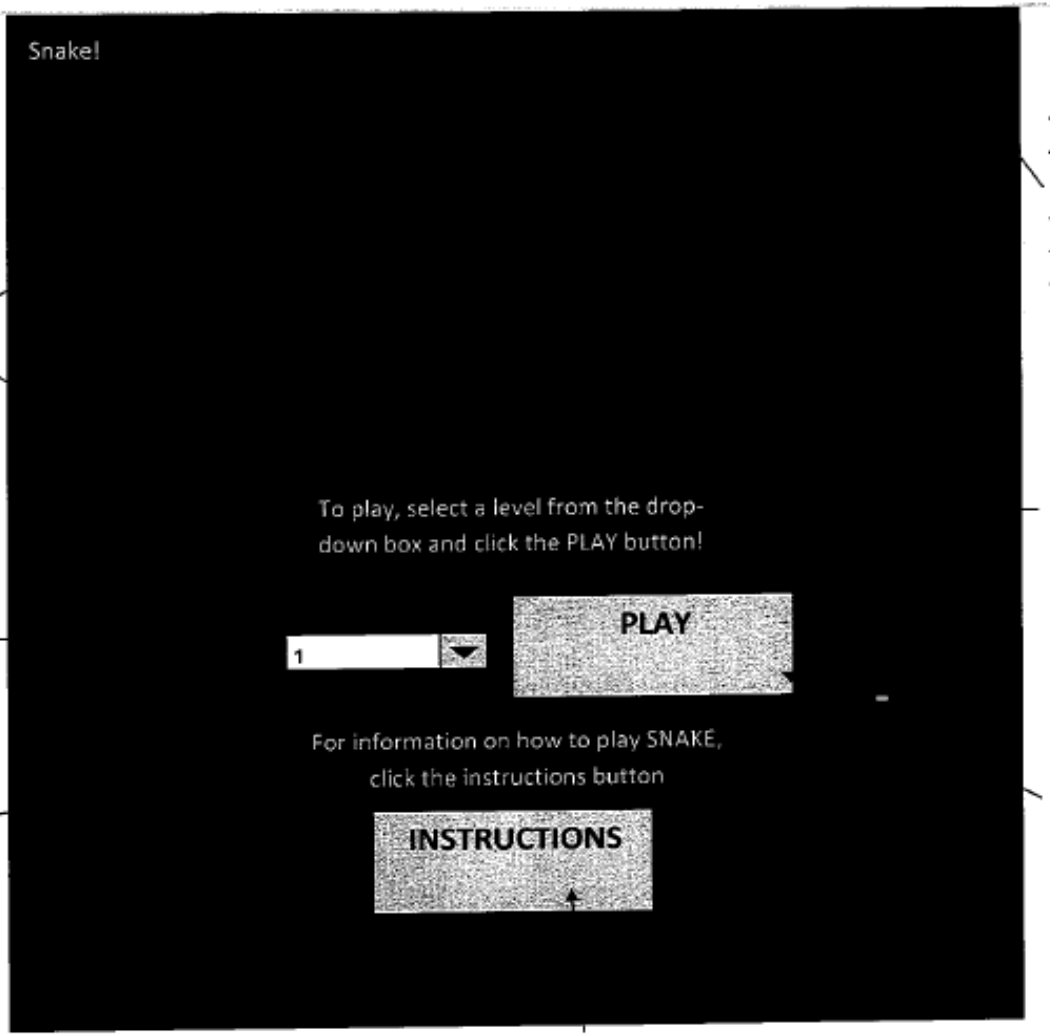
Signed:

Interface Design 1: Level Selection Screen

The initial level selection screen will be using a form named frmSnake.
The table below lists the properties of the objects particular to this screen.

Object	Property	Setting
frmSnake	BackColour	&H00000000& (Black)
	MaxButton	False
	Caption	Snake!
frmInstructions	Visible	False
cmdPlay	Caption	PLAY
cmbLevel	List	1—10
lblInstructions1	Caption	To play, select a level from the drop-down box and click the PLAY button!
	ForeColor	&H8000000E& (White)
	Font	Sans Serif size 12
lblInstructions2	Caption	For information on how to play SNAKE, click the instructions button
	ForeColor	&H8000000E& (White)
	Font	Sans Serif size 12
cmdInstructions	Caption	INSTRUCTIONS

Level Selection Screen



'NAKE' logo, mock snake and food omposed of picture boxes

Combo box named cmbLevel—llows the user to select the level at vhich they wish to begin the game, he value selected is input when :mdPlay is clicked.

The level value dictates speed and ength of the snake

Label named lblInstructions2

frmSnake.MaxButton is false, therefore maximise button not usable so that the user cannot alter what are perceived to be the borders of the game screen

Label named lblInstructions

Command button named cmdPlay—when clicked this sets the visible property of all objects shown to false, sets the level value to the value selected by the player in cmbLevel, loads and makes visible the appropriate amount of snake images and sets the value of tmrMoveSnake.interval to a value corresponding to the level chosen (tmrMoveSnake is the timer that will be used to control the snake's movements)

Command button named cmdInstructions—when clicked this sets the visible property of frmInstructions to True and the visible property of frmSnake to False

Level selection screen interview

So that a discussion can take place that results in agreement between my users, I shall conduct group interviews with all seven of my users present to ascertain the particulars of my interface design.

These are the questions that I intend to ask:

- What changes (if any) do you feel should be made to the look of the level selection screen?
- What changes (if any) do you feel should be made to the functionality of the level selection screen (e.g. with regards to command buttons and instructions)?

After each response, I shall add any further questions that I raised underlined below the response. The interview took place in front of a computer where I could alter the appearance of the interfaces during the interview to get further feedback. The final screen agreed upon is shown below.

What changes (if any) do you feel should be made to the look of the level selection screen?

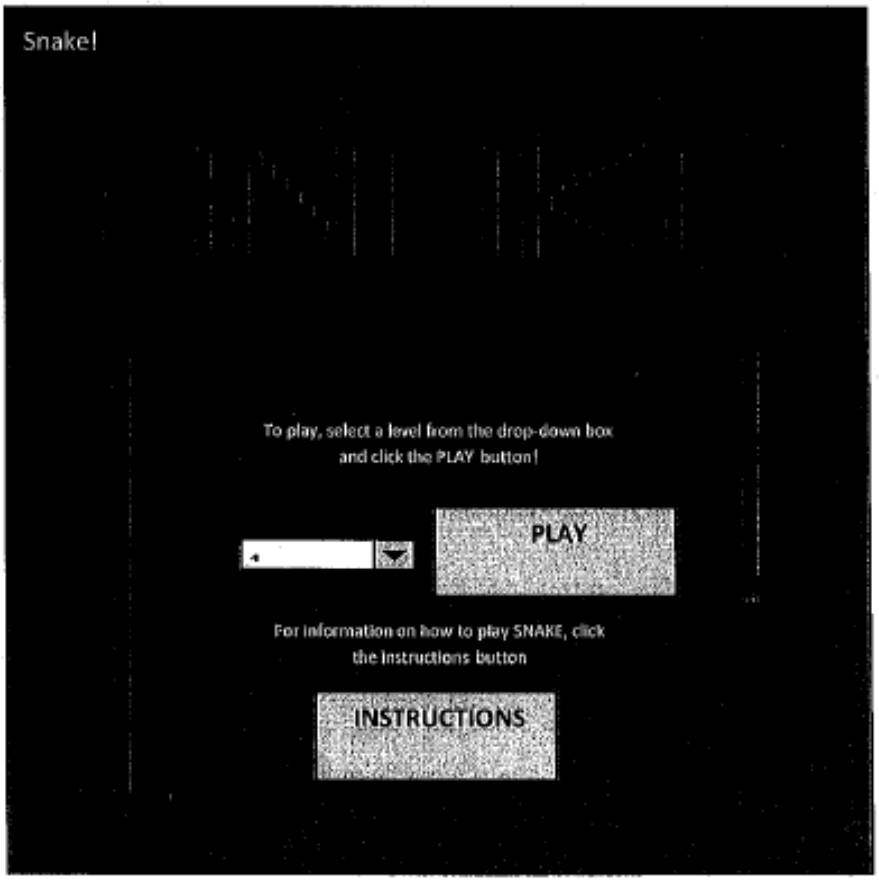
- It is too dark – a dark grey colour would be better suited to the menu screens

(After displaying backgrounds of various shades of grey) Are you happy with a background of this colour?

- Yes

What changes (if any) do you feel should be made to the functionality of the level selection screen (e.g. with regards to command buttons and instructions)?

- None



Revised row of frmSnake property table:

Object	Property	Setting
frmSnake	BackColour	&H00808080& (Dark Grey)

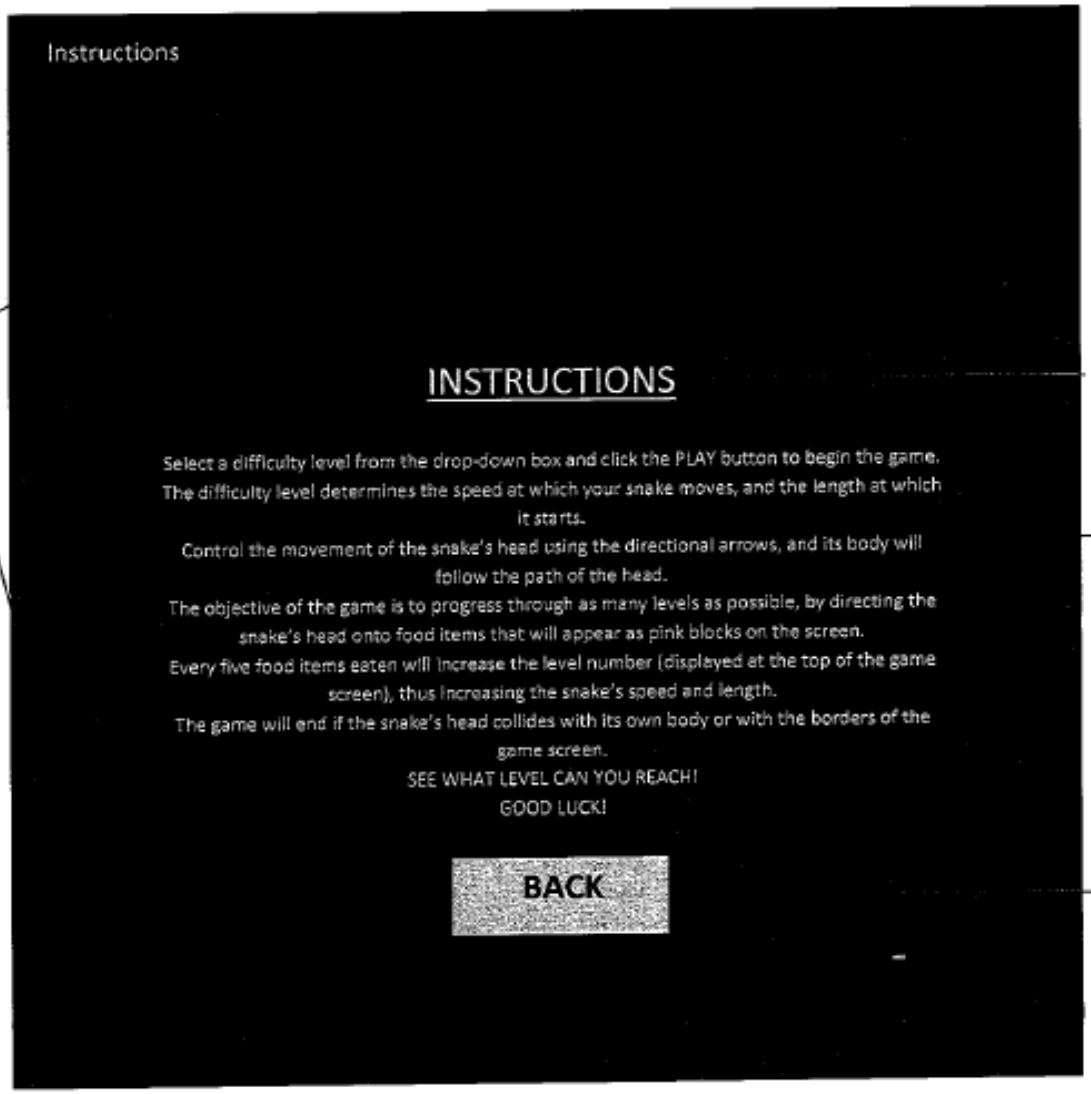
Interface Design 2: Instructions Screen

The instructions screen will be made using a form named frmInstructions.
The table below lists the properties of the objects particular to this screen.

Object	Property	Setting
frmInstructions	Visible	True
	BackColour	&H00808080& (Dark Grey)
	Caption	Instructions
frmSnake	Visible	False
cmdBack	Caption	Back
lblInstructions3	Caption	Instructions
	ForeColor	&H8000000E& (White)
	Font	Sans Serif size 20
lblInstructions4	Caption	Select a difficulty level from the drop-down box and click the PLAY button to begin the game. The difficulty level determines the speed at which your snake moves, and the length at which it starts. Control the movement of the snake's head using the directional arrows, and its body will follow the path of the head. The objective of the game is to progress through as many levels as possible, by directing the snake's head onto food items that will appear as pink blocks on the screen. Every five food items eaten will increase the level number (displayed at the top of the game screen), thus increasing the snake's speed and length. The game will end if the snake's head collides with its own body or with the borders of the game screen. SEE WHAT LEVEL CAN YOU REACH! GOOD LUCK!

Object	Property	Setting
lblInstructions4	ForeColor	&H8000000E& (White)
lblInstructions4	Font	Sans Serif size 12
cmdBack	Caption	BACK

Instructions Screen



SNAKE' logo, mock snake and food composed of picture boxes

Label named lblInstructions3

Label named lblInstructions4

Command button named cmdBack—when clicked this sets the visible property of frmInstructions to False and the visible property of frmSnake to True

Instruction screen interview

The final screen agreed upon is shown below.

What changes (if any) do you feel should be made to the look of the level selection screen?

- None

What changes (if any) do you feel should be made to the functionality of the level selection screen?

- There should be an ability to select a level and play the game without having to return to the previous screen

Additional rows of property table for frmInstructions

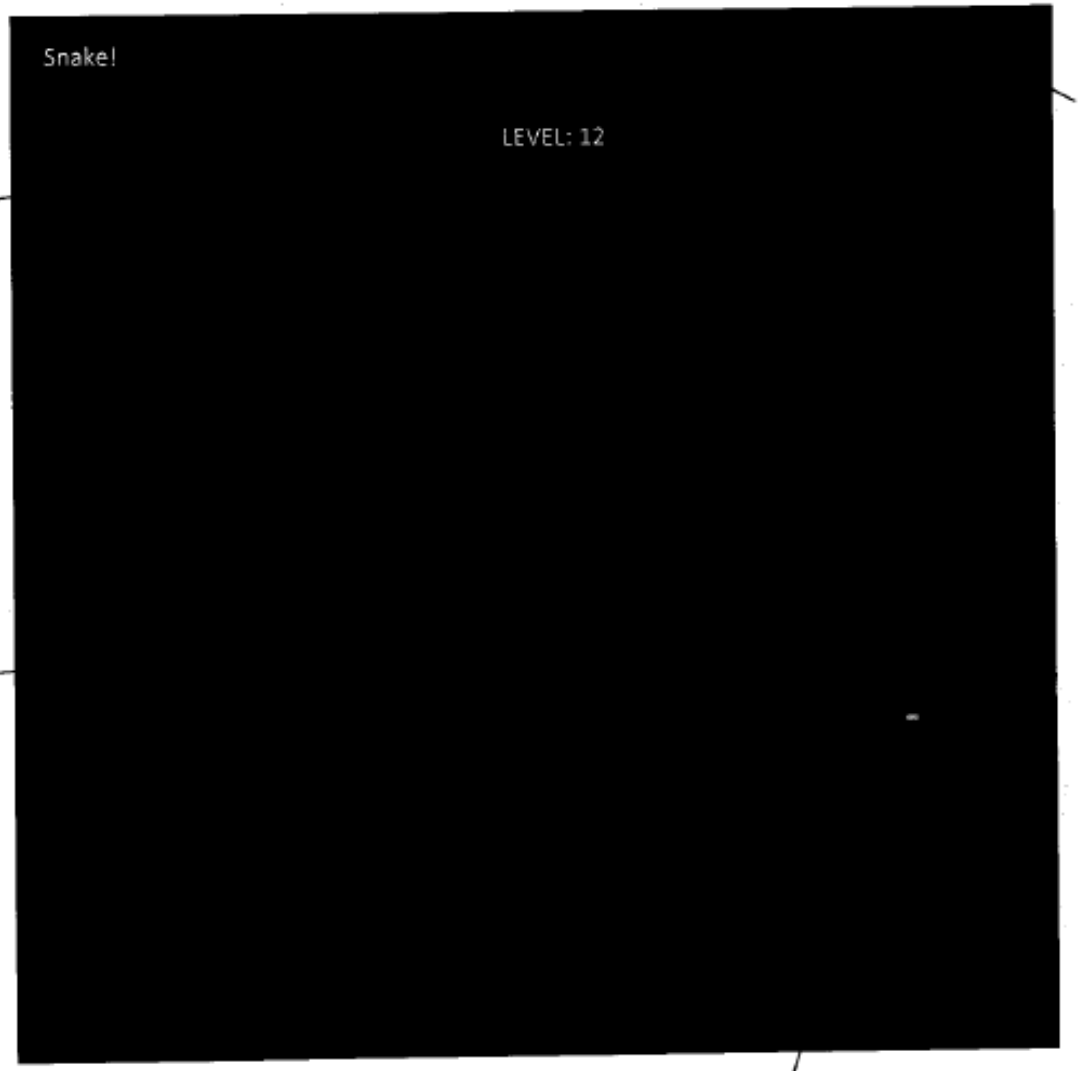
Object	Property	Setting
cmdPlay	Caption	PLAY
cmbLevel	List	1—10

Interface Design 3: Gameplay Screen

The gameplay screen also uses the form frmSnake. The below table lists the objects on this screen and their particular properties.

Object	Property	Setting
frmSnake	BackColour	&H00000000& (Black)
	MaxButton	False
	Caption	Snake!
frmInstructions	Visible	False
imgSnake	Backcolour	&H0000C000& (Green)
lblLevel	Caption	"LEVEL : " & level

Gameplay Screen



Level number displayed in a label named lblLevel, the caption property of which will be "LEVEL : " & [variable containing level number]

frmSnake.MaxButton is false, therefore maximise button not usable so that the user cannot alter what are perceived to be the borders of the game

Snake made up of an array of picture boxes named imgSnake

Food items represented by a picture box named imgFood

Gameplay screen interview

What changes (if any) do you feel should be made to the look of the gameplay screen?

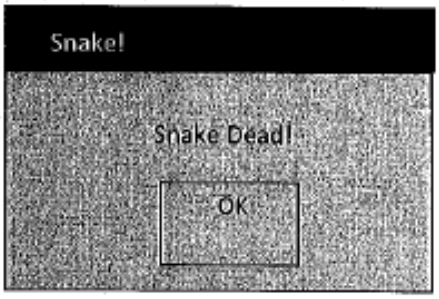
- None

What changes (if any) do you feel should be made to the functionality of the gameplay screen?

- None

Error message design

A simple error message, as follows, will be displayed when the snake collides with its own body or with the borders of the game screen.



Amendments to design objectives

In light of the user feedback obtained on the interface designs, the initial design objectives have been slightly modified. The altered and new sections are highlighted. All seven users have again read and agreed these objectives.

1) Aesthetic considerations
    a. The screen size will be large – (900x900 pixels)
    b. The colour of the snake will be: green
    c. The colour of the food will be: pink
    d. The colour of the background for the game screen will be: black
    e. The colour of the background for the menu screens will be: dark grey
    f. Each 'bit' of the snake will be 10x10 pixels in size
    g. The snake should start at a length of thirty 'bits' on level 1
    h. The snake's head should initially be facing right
    i. Each food item will be 10x10 pixels in size
    j. The time interval between each snake movement at level 1 will be 100 milliseconds
    k. There is no pre-defined limit on the number of levels to be played through (instead this is limited by the skill of the player)

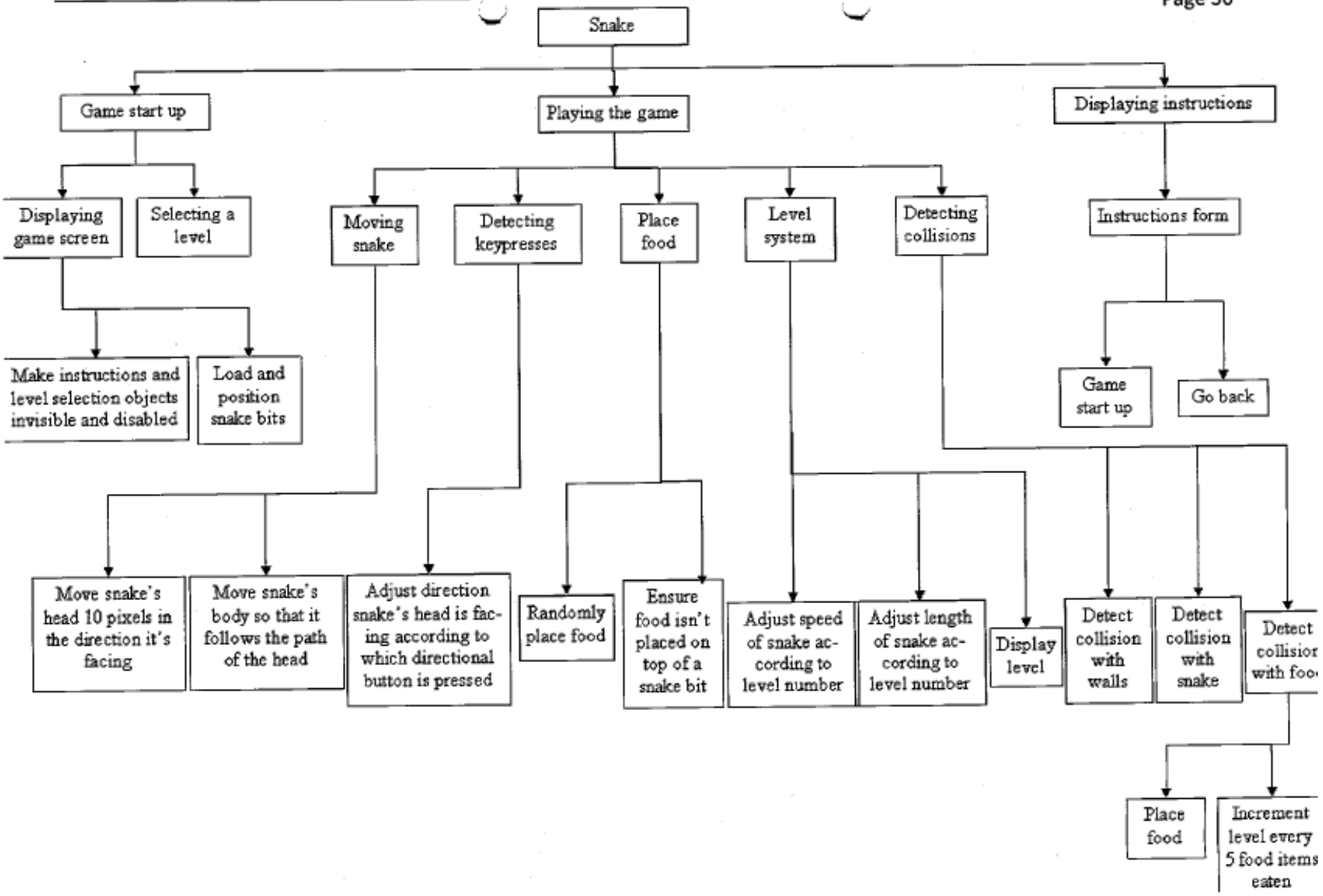
2) Input considerations
   a. The player will be able to choose the level at which he/her begins the game from (between 1 and 10) with the use of a drop down list and combo box on both the initial menu screen and the instructions screen
   b. The player will be able to control the direction that the head is facing with the use of the directional keys on a keyboard
   c. The player will not be able to make the snake go back on itself by pressing left when the head is facing right, or up when the head is facing down etc.
   d. The player will be able to view an instructions screen that opens in a new form by clicking a command button that is on the initial menu screen
   e. The player will be able to go back to the initial level selection screen from the instructions screen with the use of a command button
3) Processing considerations
   a. The coordinates of the snake's head will be continually incremented by 10 pixels in the direction that the snake's head is facing at a time interval that is determined by the level chosen
   b. Each snake bit behind the head will be incremented by 10 pixels in the direction that the snake's head was facing on a previous move (with the move number corresponding to the snake bit's distance from the head, such that the snake's body will follow the path of the head)
   c. It must be determined with each movement whether the snake's head collides into its body, in which case an error message should be produced and the game should end
   d. It must be determined with each movement whether the snake's head collides into the borders of the game screen, in which case an error message should be produced and the game should end
   e. It must be determined with each movement whether the snake's head collides into a food item, in which case the a the food item will be randomly moved to another position on the game screen
   f. The system must record the amount of food items eaten, and increment the level number every time 5 food items are eaten
   g. Up to and including the transition to level 10, the time intervals between each movement of the snake must be decreased by 10 milliseconds every time the level number is increased. This will be done using a timer to control the snake movements.
   h. At each new level the length of the snake must be increased by 6 bits
4) Output considerations
   a. The current level number will be displayed
   b. The positions of each snake bit will be displayed in real time
   c. The position of the food item will be displayed
   d. Instructions on how to select a level be displayed on the initial level selection screen
   e. Instructions on how to play the game will be displayed in a new form when the instructions command button is clicked

User review of objectives

All seven of my users have reviewed and agreed to the above objectives.

Signed:

Modular Design of Snake – Structure Diagram



Data Structure Design

No files will be created or used by the Snake program, however the program will, of course, make use of variables. The data type 'SnakeBit' is a user defined type that will be defined in a public module named mdlSnake.
SnakeBit consists of the following variables:

Variable Name	Type	Size (range of possible values)	Description	Sample Value	Validation
Direction	Byte	1 - 4	Contains the direction that an individual snake bit is facing (1 = down, 2 = right, 3 = up, 4 = left)	2	NA
X	Long	-10 - 900	Contains the X co-ordinate of the left of an individual snake bit (in pixels)	20	If X >= 900 or X < 0 then the game ends
Y	Long	-10 - 900	Contains the Y co-ordinate of the top of an individual snake bit (in pixels)	600	If Y >= 900 or Y < 0 then the game ends

Below are the descriptions of the variables that are to be used by the program:

Variable Name	Type	Size (for numeric types this is the range of possible values, for alphanumeric types this is the length of the string)	Description	Sample Value	Validation
snake(0 to 9999)	SnakeBit	Different for each of snake(n).Direction, snake(n).X and snake(n).Y – see above	Stores the X and Y values, and direction facing for each snake bit	Different for each of snake(n).Direction, snake(n).X and snake(n).Y – see above	See above
level	Byte	1 - 255	Stores the value of the level selected (this determines the length of the snake and of timer intervals)	12	NA
valid	Boolean	0(FALSE) – 1(TRUE)	Used to flag up if a food item is to be placed on top of a snake bit so that this can be avoided	1(TRUE)	NA
Variable Name	Type	Size (for numeric types this is the range of possible values,	Description	Sample Value	Validation

		for alphanumeric types this is the length of the string)			
length	Integer	0 - 9999	Stores the length of the snake and is equal to the number of snake bits minus 1	22	NA
foodTop	Single	0 - 890	Contains the Y co-ordinate of the top of a food item (in pixels)	500	FoodTop and foodLeft cannot have the same values as the Y and X values respectively of any snake bit (because then the food would be placed on top of the snake)
foodLeft	Single	0 – 890	Contains the X co-ordinate of the left of a food item (in pixels)	490	Same as validation for foodTop
facing(-84 To 999999)	Byte	1 - 4	Stores the direction the head was facing (the value stored) on a particular move (the index)	3	NA
moves	Long	0 - 999999	Stores amount of times the timer is called	8883	NA
keyPressed	Single	1 - 145	Stores the keycode value of any button that is pressed	37	NA
foodCount	Byte	0 - 4	Stores how many food items are eaten, up to every 5th item eaten	2	NA

Test strategy – Test plan

In the testing of Snake, I shall first perform white-box testing in the form of dry runs to be carried out on the design algorithms on page 37. Secondly, I shall perform black-box testing on the finished program, using the test plan outlined below, and respond accordingly to any issues that present themselves. Finally I shall have my user group test the program and give feedback, which may result in the modification of the program.

The Snake program does not lend itself to the input of 'extreme' data, except perhaps when the snake traverses the very edges of the game screen. This shall be tested by confirming that the game ends the *instant* the snake goes beyond the borders of the game screen, and not before. Each test shall be evidenced with screenshots.

Function being tested	Method of input	Input	Expected result
Level selection	Test 1: Combo-box (cmbLevel) and command button (cmdLevel) on menu screen Test 2: Combo box (cmbLevel) and command button (cmdLevel) on instructions screen	1	Snake length: 30 bits long Snake facing: right Timer interval: 100 milliseconds lblLevel caption: "LEVEL: 1"
		2	Snake length: 36 bits long Snake facing: right Timer interval: 90 milliseconds lblLevel caption: "LEVEL: 2"
		3	Snake length: 42 bits long Snake facing: right Timer interval: 80 milliseconds lblLevel caption: "LEVEL: 3"
		4	Snake length: 48 bits long Snake facing: right Timer interval: 70 milliseconds lblLevel caption: "LEVEL: 4"
		5	Snake length: 54 bits long Snake facing: right Timer interval: 60 milliseconds lblLevel caption: "LEVEL: 5"

Level selection	Test 1: Combo-box (cmbLevel) and command button (cmdLevel) on menu screen Test 2: Combo box (cmbLevel) and command button (cmdLevel) on instructions screen	6	Snake length: 60 bits long Snake facing: right Timer interval: 50 milliseconds lblLevel caption: "LEVEL: 6"
		7	Snake length: 66 bits long Snake facing: right Timer interval: 40 milliseconds lblLevel caption: "LEVEL: 7"
		8	Snake length: 72 bits long Snake facing: right Timer interval: 30 milliseconds lblLevel caption: "LEVEL: 8"
		9	Snake length: 78 bits long Snake facing: right Timer interval: 20 milliseconds lblLevel caption: "LEVEL: 9"
		10	Snake length: 84 bits long Snake facing: right Timer interval: 10 milliseconds lblLevel caption: "LEVEL: 10"
Movement of snake	When in a game, pressing a directional button on the keyboard	Up (when head is facing right or left)	Snake's head will face and therefore move upwards, and each trailing snake bit will move to follow the path of the head every time the timer is called
		Right (when head is facing up or down)	Snake's head will face and therefore move right, and each trailing snake bit will move to follow the path of the head every time the timer is called
		Down (when head is facing right or left)	Snake's head will face and therefore move down, and each trailing snake bit will move to follow the path of the head every time the timer is called
		Left (when head is facing up or down)	Snake's head will face and therefore move left, and each trailing snake bit will move to follow the path of the head every time the timer is called
		Up (when head is facing up or down)	No change

Function being tested	Method of input	Input	Expected result
Movement of snake	When in a game, pressing a directional button on the keyboard	Right (when head is facing right or left)	No change
		Down (when head is facing up or down)	No change
		Left (when head is facing right or left)	No change
		None	Coordinates of snake's head are continually incremented by 10 pixels in the direction that the snake's head is facing and each trailing snake bit moves to follow the path of the head every time the timer is called
Collision with walls	When in a game, directing snake's head into a border of the game screen using directional buttons on the keyboard	Up, to direct snake's head into top border of game screen	The instant the snake's head goes beyond the border of the game screen, an error message is produced and the game ends, returning to the initial menu screen
		Right, to direct snake's head into right border of game screen	
		Down, to direct snake's head into bottom border of game screen	
		Left, to direct snake's head into left border of game screen	
Collision with own body	When in a game, directing snake's head into its own body using directional buttons on the keyboard	Directional buttons on keyboard used to direct snake's head	The instant the snake's head has the same coordinates as any one of its trailing snake bits, an error message is produced and the game ends, returning to the initial menu screen
Collision with food (with no level advancement)	When in a game, directing snake's head into a food item using directional buttons on the keyboard	Directional buttons on keyboard used to direct snake's head into a food item (excluding every fifth food item eaten)	Food item randomly moved to another position on the game screen (that is not currently occupied by a snake bit)

Function being tested	Method of input	Input	Expected result
Collision with food (with level advancement)	When in a game, directing snake's head into a food item using directional buttons on the keyboard	Snake directed onto a food item when foodCount = 4, and level number is <= 10	Food item randomly moved to another position on the game screen (that is not currently occupied by a snake bit); Snake length: increased by 6 bits; Timer interval controlling snake speed: decreased by 10 milliseconds; lblLevel caption: "LEVEL: [level]"
		Snake directed onto a food item when foodCount = 4, and level number is > 10	Food item randomly moved to another position on the game screen (that is not currently occupied by a snake bit); Snake length: increased by 6 bits; lblLevel caption: "LEVEL: [level]" No change to timer interval
Unlimited level advancement	Disable collision detection with own body or borders of game screen, then continually advance through levels up to level 50	NA	Up to and including level 10, the interval of the timer controlling the snake's speed will be decreased by 10 at every new level and snake length will be increased by 6 bits; Beyond level 10, the timer interval will remain at 10 milliseconds, and at every new level the snake length will be increased by 6 bits
Displaying instructions	Clicking the instructions button (cmdInstructions) on the menu screen	NA	Displays instructions screen by setting the visible property of frmInstructions to True and the visible property of frmSnake to False
Returning to initial menu screen from the instructions screen	Clicking the back button (cmdBack) on the instructions screen	NA	Returns to initial menu screen by setting the visible property of frmInstructions to False and the visible property of frmSnake to True

All seven members of my user group have seen and agreed to the above test strategy and modular design structure diagram.

Signed:

Algorithms & Testing

The snake program will require several algorithms within various modules. The module names are written below in italics (these can be seen to correspond with the modules specified in the structure diagram on page 30) and the algorithms in each module are bulleted below them. I shall describe each algorithm in turn using pseudo-code.

In order to test each algorithm for functionality, I have performed dry runs of each algorithm that is more complex than a simple sequence - the results of which are displayed below the algorithm.

1) *Game start up*
    - Make instructions and level selection objects invisible and disabled
    - Load and position snake bits
2) *Placing food*
    - Randomly place food and ensure that food isn't placed on top of a snake bit
3) *Detecting keypresses*
    - Adjust direction snake's head is facing according to which directional button is pressed, but not allowing the snake to go back on itself
4) *Detecting collisions*
    - Detect collision with walls
    - Detect collision with snake
    - Display initial menu screen if collision with walls or snake is detected
    - Detect collision with food, place new food and level up if necessary
5) Moving snake
    - Move snake's body so that it follows the path of the head
    - Move snake's head 10 pixels in the direction it's facing and store the move number on which the snake's head is facing a particular direction

1) Game start up

Make instructions and level selection objects invisible and disabled
(This is written as a simple sequence, as part the start-up subroutine called StartGame)

Make lblInstructions invisible
Make cmdLevel invisible
Make cmbLevel invisible
Make cmdInstructions invisible
Disable cmdLevel
Disable cmbLevel
Disable cmdInstructions
Change back colour of frmSnake to black

Load and position snake bits
(This should load and position the appropriate amount of snake bits according to the level that is selected, assuming that the program has 5 snake bits loaded by default. It should also ensure that all loaded snake bits are facing right, and it also sets the corresponding X and Y values of the associated snake variable accordingly. This will also be part of the start up subroutine called StartGame.)

```
01  length = 30 + ((level - 1) * 6)
02  FOR k = 0 TO (length - 1)
03      IF k > 4 THEN Load imgSnake(k)
04      Make imgSnake(k) visible
05      Top value of imgSnake(k) = 300
06      snake(k).Y = 300
07       Left value of imgSnake(k) = 300 - (10 * k)
08       snake(k).X = 300 - (10 * k)
09       snake(k).Direction = 2
10  NEXT k
```

For the following dry run I shall have the level set to 3 and shall end the dry run after 2 successful runs of lines 02 to 10, as this should adequately test the algorithm.

Lines	level	length	k	snake().Y	snake().X	snake().direction	Output	Comment
01	3	42						
02			0					
03								FALSE, go to line 04
04							imgSnake(0) becomes visible	
05							imgSnake(0) gets assigned a top value of 300	
06				snake(0).Y = 300				

07							imgSnake(0) gets a left value of 300	
08					snake(0).X = 300			
09						snake(0).direction =2		
10, 02			1					
03								FALSE, go to line 04
04							imgSnake(1) becomes visible	
05							imgSnake(1) gets assigned a top value of 300	
06				snake(1).Y = 300				
07							imgSnake(1) gets a left value of 290	
08					snake(1).X = 290			
09						snake(1).direction =2		
10, 02			2					

2) Placing food

Randomly places food and ensures that food isn't placed on top of a snake bit
(This will be written in a subroutine called PlaceFood)

```
01  DO
02    Randomize the random function
03    valid = TRUE
04    foodTop = The integer value of (a random number * 89) * 10
05    foodLeft = The integer value of (a random number * 89) * 10
06    FOR k = 0 TO (length - 1)
07      IF foodTop =Top value of snake(k).Y AND foodLeft = Left value of Snake(k).X THEN
08        valid = FALSE
09        Exit the for...next loop
10      END IF
11    NEXT k
12  LOOP UNTIL valid = TRUE
13  Make imgFood Visible
14  Make top value of imgFood = foodTop
15  Make left value of imgFood = foodLeft
```

In order to perform a dry run that fully tests this algorithm, I shall write one trace table where the arbitrarily chosen 'random' numbers that dictate the food's coordinates are such that they would cause the food to be placed on top of a snake bit represented by imgSnake(1), and one where there is no such issue. In both tables, I shall assume that the snake is at starting position and that it is at a length of 5 bits.

Lines	valid	arbitrarily chosen random number	foodTop	foodLeft	k	snake(k).Y	snake(k).X	Output	Comment
01, 02									Random function is randomized
03	TRUE								Initialises valid
04		0.3379564	300						
05		0.3283333		290					
06					0				
07						300	300		FALSE, go to line 10
10, 11, 06					1				
07						300	290		TRUE, go to line 08
08	FALSE								
09, 12, 01,									Random

02									function is randomized
03	TRUE								
04		0.666221	590						
05		0.442133		390					
06					0				
07						300	300		FALSE, go to line 10
10, 11, 06					1				
07						300	290		False, go to line 10
10, 11, 06					2				
07						300	280		False, go to line 10
10, 11, 06					3				
07						300	270		False, go to line 10
10, 11, 06					4				
07						300	260		False, go to line 10
10, 11, 12									
13								imgFood becomes visible	
14, 15								imgFood is moved to the coordinates foodTop [X] and foodLeft [Y]	

Lines	valid	arbitrarily chosen random number	foodTop	foodLeft	k	snake(k).Y	snake(k).X	Output	Comment
01, 02									Random function is randomized
03	TRUE								Initialises valid
04		0.293439	260						
05		0.583493		510					
06					0				
07						300	300		FALSE, go to line 10
10, 11, 06					1				
07						300	290		False, go to line 10

10, 11, 06					2				
07						300	280		False, go to line 10
10, 11, 06					3				
07						300	270		False, go to line 10
10, 11, 06					4				
07						300	260		False, go to line 10
10, 11, 12									
13								imgFood becomes visible	
14, 15								imgFood is moved to the coordinates foodTop [X] and foodLeft [Y]	

3) Detecting keypresses

Adjust direction snake's head is facing according to which directional button is pressed, but not allowing the snake to go back on itself
(This will be written in a subroutine that is called when a key is pressed on the keyboard. I shall use the key down event for forms in visual basic.)

```
keyPressed = Key code value of the button pressed
SELECT CASE keyPressed
   CASE [Key code for left directional button]
   IF snake(0).Direction = 1 OR snake(0).Direction = 3 THEN snake(0).Direction = 4
   CASE [Key code for up directional button]
   IF snake(0).Direction = 2 OR snake(0).Direction = 4 THEN snake(0).Direction = 3
   CASE [Key code for right directional button]
   IF snake(0).Direction = 1 OR snake(0).Direction = 3 THEN snake(0).Direction = 2
   CASE [Key code for down directional button]
   IF snake(0).Direction = 2 OR snake(0).Direction = 4 THEN snake(0).Direction = 1
END SELECT
```

4) Detecting collisions

Detect collision with walls
(This will be written in a subroutine called CheckWallCollision)

```
IF snake(0).X >= 900 OR snake(0).X < 0 OR snake(0).Y >= 900 OR snake(0).Y < 0 THEN
  Disable tmrMoveSnake
  OUTPUT Error message
  FOR i = 0 TO (length – 1)
    Make imgSnake(i) invisible
  NEXT i
  CALL EndGame Subroutine
END IF
```

Detect collision with snake
(This will be written in a subroutine called CheckSnakeCollision)

```
FOR k = 1 To (length - 1)
  IF snake(0).Y = snake(k).Y AND snake(0).X = snake(k).X THEN
    Disable tmrMoveSnake
    OUTPUT Error message
    FOR i = 0 TO (length - 1)
      Make imgSnake(i) invisible
    NEXT i
    CALL EndGame Subroutine
  End If
NEXT k
```

Display initial menu screen if collision with walls or snake is detected
(This will be written in a subroutine called EndGame)

```
Change background colour of frmSnake to dark grey
Make imgFood invisible
Make lblInstructions visible
Make lblInstructions2 visible
Make cmdLevel visible
Make cmbLevel visible
Enable cmdLevel
Enable cmbLevel
Make cmdInstructions visible
Enable cmdInstructions
Make lblLevel invisible
```

Detect collision with food, place new food and level up if necessary. Set the position and direction of new bits added so that they correspond with the snake's final bit.
(This will increase level, snake length and snake speed if appropriate. It will be written in a subroutine called CheckFoodCollision.)

```
IF snake(0).Y = foodTop And snake(0).X = foodLeft THEN
  IF foodCount <> 4 THEN
    foodCount = foodCount + 1
  ELSE
    foodCount = 0
    level = level + 1
    Set the caption of lblLevel to "LEVEL: [value stored in level]"
    IF level <= 10 THEN Make the interval of tmrMoveSnake = 110 - (level * 10)
    FOR k = length TO (length + 5)
      Load imgSnake(k)
      Make imgSnake(k) Visible
      SELECT CASE snake(length - 1).Direction
        CASE 1
          Top value of imgSnake(k) = snake(k - 1).Y - 10
          snake(k).Y = Top value of imgSnake(k)
          Left Value of imgSnake(k) = (snake(k - 1).X)
          snake(k).X = imgSnake(k).Left
          snake(k).Direction = 1
          facing(moves - k) = snake(k).Direction
        CASE 2
          Top value of imgSnake(k)  = (snake(k - 1).Y)
          snake(k).Y = imgSnake(k).Top
          Left Value of imgSnake(k) = snake(k - 1).X - 10
          snake(k).X = imgSnake(k).Left
          snake(k).Direction = 2
          facing(moves - k) = snake(k).Direction
        CASE 3
          Top value of imgSnake(k) = (snake(k - 1).Y) + 10
          snake(k).Y = imgSnake(k).Top
          Left Value of imgSnake(k) = snake(k - 1).X
          snake(k).X = imgSnake(k).Left
          snake(k).Direction = 3
          facing(moves - k) = snake(k).Direction
        CASE 4
          Top value of imgSnake(k) = snake(k - 1).Y
          snake(k).Y = imgSnake(k).Top
          Left Value of imgSnake(k) = (snake(k - 1).X + 10)
          snake(k).X = imgSnake(k).Left
          snake(k).Direction = 4
          facing(moves - k) = snake(k).Direction
      END SELECT
    NEXT k
```

```
      length = length + 6
    END IF
    CALL PlaceFood Subroutine
  END IF
```

5) Moving Snake

Move snake's body so that it follows the path of the head
(This moves the snake's body by 10 pixels according to the direction that the head was facing on the move numbered (moves - number of snake bit). It will be written in a subroutine that is called at each interval of tmrMoveSnake, named MoveSnakeBody)

```
01  FOR i = 1 TO (length - 1)
02    snake(i).Direction = facing(moves - i)
03    SELECT CASE snake(i).Direction
04      CASE 1
05      snake(i).Y = snake(i).Y + 10
06      Top value of imgSnake(i) = snake(i).Y
07      CASE 2
08      snake(i).X = snake(i).X + 10
09      Left value of imgSnake(i) = snake(i).X
10      CASE 3
11      snake(i).Y = snake(i).Y - 10
12      Top value of imgSnake(i) = snake(i).Y
13      CASE 4
14      snake(i).X = snake(i).X - 10
15      Left value of imgSnake(i) = snake(i).X
16    END SELECT
17  NEXT i
```

Move snake's head 10 pixels in the direction it's facing and store the move number on which the snake's head is facing a particular direction
(This will be written in a subroutine called MoveSnakeHead.)

```
SELECT CASE snake(0).Direction
  CASE 1
  snake(0).Y = snake(0).Y + 10
  Top value of imgSnake(0) = snake(0).Y
  facing(moves) = 1
  CASE 2
  snake(0).X = snake(0).X + 10
  Left value of imgSnake(0) = snake(0).X
  facing(moves) = 2
  CASE 3
  snake(0).Y = snake(0).Y - 10
  Top value of imgSnake(0) = snake(0).Y
  facing(moves) = 3
```

```
    CASE 4
    snake(0).X = snake(0).X - 10
    Left value of imgSnake(0) = snake(0).X
    facing(moves) = 4
  END SELECT
```

For the sake of clarity, I shall now describe the sequence in which the aforementioned subroutines are called.

01) StartGame
02) PlaceFood
03) MoveSnakeBody
04) MoveSnakeHead
05) CheckWallCollision -If a wall collision is detected then EndGame is called, returning the user to the initial level selection screen. If not then...
06) CheckSnakeCollision - If a snake collision is detected then EndGame is called, returning the user to the initial level selection screen. If not then...
07) CheckFoodCollision – If a food collision is detected then PlaceFood is called, and the cycle continues from there. If not then MoveSnakeBody is called, and the cycle continues from there.

If at any point a key is pressed then the Key Down event is executed after which MoveSnakeBody is called, and the cycle continues from there.

Section 3 – Software development and testing

Part 1 – Software development.

The software development process can be broken down into several key stages, which I shall describe in the order in which they were carried out - highlighting the problems that were encountered along the way, and how they were resolved.

1) Begin writing StartGame subroutine and make basic interface

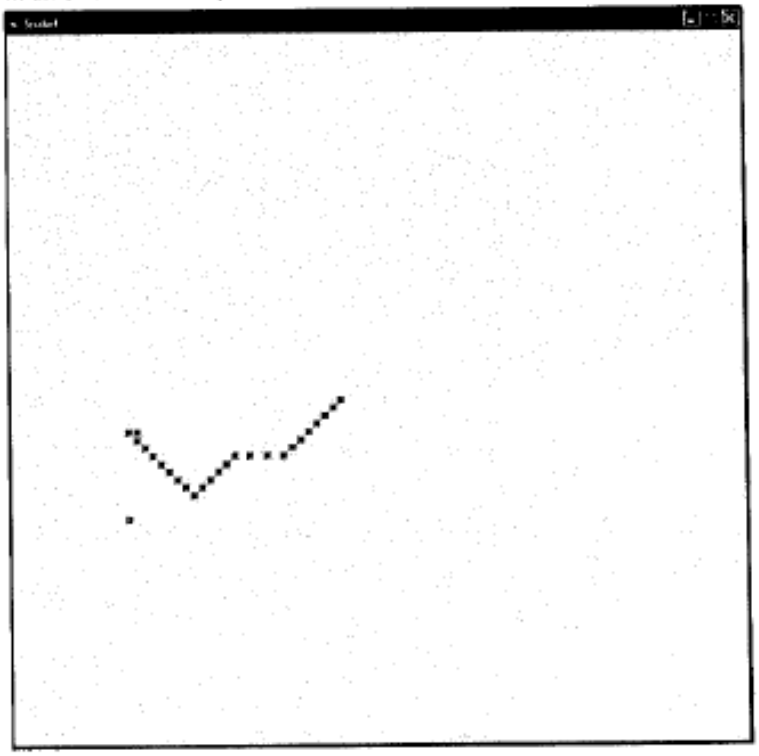
Firstly I made a basic user interface including the command button (cmdLevel) and list box (cmbLevel) needed to select a level and play the game, five image boxes in a control array named imgSnake, and the timer used to control snake movement (tmrMoveSnake). I then began writing the StartGame subroutine that is executed when cmdLevel is clicked, firstly by writing the code to load the game screen, and then by writing the code that loads and positions the appropriate number of snake bits, according to which level is selected.

2) Write MoveSnakeHead subroutine and the subroutine for detecting key presses

At this point, alpha testing demonstrated that the snake's head was able to move around the screen independently of its body, and it was possible to control the snake's movement with the directional arrows on the keyboard.

3) Write MoveSnakeBody subroutine

I wrote the MoveSnakeBody subroutine, and the snake's body did then follow the path of the head – but it did so in an erratic manner, as can be seen below:



In order to determine the cause of this error, I put a breakpoint at the beginning of the StartGame subroutine and ran the program. After stepping through the program, when I reached the timer call, I discovered that the variable 'moves', that stores the amount of moves made by the snake's head so that the trailing bits can face the direction the head was facing on a particular move, was not at the correct starting value of 0. I then realised the cause of the error: the timer had been running since the loading of the program. This was fixed by simply setting the default value of tmrMoveSnake.Enabled to false, and changing it to true when cmdLevel is clicked with the line:

```
tmrMoveSnake.Enabled = True
```

4) Write PlaceFood subroutine

Food is now placed randomly on the game screen, and does not seem to be placed on top of a snake bit.

5) Write CheckWallCollision and EndGame subroutines

The snake is no longer able to go beyond the borders of the game screen, and the appropriate error message is displayed when it does so, and the initial level selection screen is displayed.

6) Write CheckSnakeCollision subroutine

The snake can no longer travel through itself – upon collision, the appropriate error message and the initial level selection screen are displayed.

At this point in development, alpha testing demonstrated that the snake's body followed the path of the head as desired, and at most times it was not possible to make the snake go back on itself – however, if two directional buttons were pressed within X milliseconds of each other (where X = the interval of tmrMoveSnake at the level being played) then it was possible to make the snake move back on itself, thus losing the game. This problem was fixed by changing the subroutine that detects keypresses, as below:

Old code

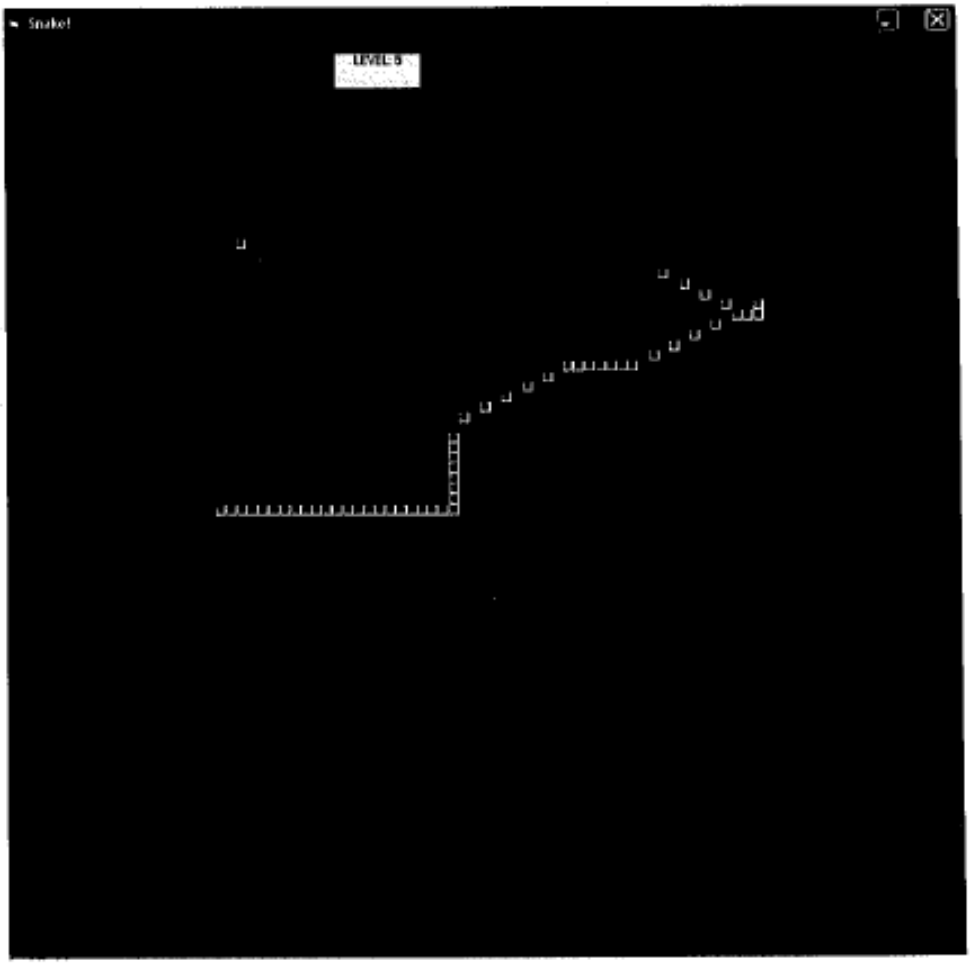
```
keyPressed = KeyCode
Select Case keyPressed
   Case 37
   If snake(0).Direction = 1 Or snake(0).Direction = 3 Then snake(0).Direction = 4
   Case 38
   If snake(0).Direction = 2 Or snake(0).Direction = 4 Then snake(0).Direction = 3
   Case 39
   If snake(0).Direction = 1 Or snake(0).Direction = 3 Then snake(0).Direction = 2
   Case 40
   If snake(0).Direction = 2 Or snake(0).Direction = 4 Then snake(0).Direction = 1
End Select
```

New code

```
keyPressed = KeyCode
Select Case keyPressed
   Case 37
   If snake(1).Y <> snake(0).Y Or snake(1).Y <> snake(0).Y Then snake(0).Direction = 4
   Case 38
   If snake(1).X <> snake(0).X Or snake(1).X <> snake(0).X Then snake(0).Direction = 3
   Case 39
   If snake(1).Y <> snake(0).Y Or snake(1).Y <> snake(0).Y Then snake(0).Direction = 2
   Case 40
   If snake(1).X <> snake(0).X Or snake(1).X <> snake(0).X Then snake(0).Direction = 1
End Select
```

7) Write CheckFoodCollision subroutine

At this point in development, it was possible to direct the snake onto food items, and to level up after 5 food items were eaten – decreasing the timer interval and increasing the snake's length as desired. However, the new snake bits added at each level acted erratically, as shown below.



After inserting a break point at the beginning of the part of CheckFoodCollision that is concerned with level progression and stepping through the program, I discovered that the cause of the peculiar movement of new snake bits was due to the fact that they were having their directions altered by the MoveSnakeBody subroutine, instead of retaining the direction that the snake's tail was facing.

To overcome this, I needed to ensure that the new snake bits only had their directions controlled by MoveSnakeBody after they had made their first movement. I did this by creating a string variable named newBits, which stores the index of each new bit added at a level increase in CheckFoodCollision with the line:

```
newBits = newBits & " " & k
```

Where k = the index of the new bit being added.

I then changed the MoveSnakeBody subroutine so that the direction of a snake bit was only altered if it was not a new bit, and so that the newBits string would be emptied once each new bit had been moved once. I did this by changing the single line:

```
snake(i).Direction = facing(moves - i)
```

To the following:

```
If InStr(newBits, " " & i & " ") = 0 Then
        snake(i).Direction = facing(moves - i)
End If
If i = length - 1 Then newBits = ""
```

Where i = the index of the snake bit being moved.

8) Create instructions form

Upon creating the instructions form, there was an issue whereupon nothing happened when I tried to start a new game from the instructions form. This was due to the value of the level selected in cmbLevel not being transferred to the level variable. I resolved this issue by declaring level as a public variable in the module named mdlSnake.

9) Create user interfaces that match the designs

Using many picture boxes to display the snake logos on the instructions and level selection screens caused the program to slow down. This issue was resolved by taking a screenshot of the logos and replacing the many individual picture boxes with single picture boxes containing images of the logos, named imgLogo1, imgLogo2, and imgLogo3. Also, the labels used in frmSnake to display instructions (lblInstructons and lblInstructions2) were then hidden behind the picture box used to display the logo. This problem could not be resolved by changing the order properties of the objects concerned, because picture boxes always appear in front of labels in VB 6. Instead, I placed lblInstructions and lblInstructions2 in picture boxes of their own. The program is now essentially complete, according to the design specification outlined and agreed to on page 16.

Part 2 –Testing

I shall now test the finished program according to the test strategy outlined on page 33. Each test has been demonstrated by screenshots that are displayed below the test results.

Function being tested	Method of input	Input	Expected result	Actual result	Screenshot reference
Level selection	Test 1: Combo-box (cmbLevel) and command button (cmdLevel) on menu screen Test 2: Combo box (cmbLevel) and command button (cmdLevel) on instructions screen	1	Snake length: 30 bits long Snake facing: right Timer interval: 100 milliseconds lblLevel caption: "LEVEL: 1"	Test 1: As expected Test 2: As expected	1 (Test 1 result is identical to test 2 result, therefore only one screenshot is displayed)
		2	Snake length: 36 bits long Snake facing: right Timer interval: 90 milliseconds lblLevel caption: "LEVEL: 2"	Test 1: As expected Test 2: As expected	2 (Test 1 result is identical to test 2 result, therefore only one screenshot is displayed)
		3	Snake length: 42 bits long Snake facing: right Timer interval: 80 milliseconds lblLevel caption: "LEVEL: 3"	Test 1: As expected Test 2: As expected	3 (Test 1 result is identical to test 2 result, therefore only one screenshot is displayed)
		4	Snake length: 48 bits long Snake facing: right Timer interval: 70 milliseconds lblLevel caption: "LEVEL: 4"	Test 1: As expected Test 2: As expected	4 (Test 1 result is identical to test 2 result, therefore only one screenshot is displayed)
		5	Snake length: 54 bits long Snake facing: right Timer interval: 60 milliseconds lblLevel caption: "LEVEL: 5"	Test 1: As expected Test 2: As expected	5 (Test 1 result is identical to test 2 result, therefore only one screenshot is displayed)
		6	Snake length: 60 bits long Snake facing: right	Test 1: As expected Test 2: As	6 (Test 1 result is identical to

			Timer interval: 50 milliseconds lblLevel caption: "LEVEL: 6"	expected	test 2 result, therefore only one screenshot is displayed)
		7	Snake length: 66 bits long Snake facing: right Timer interval: 40 milliseconds lblLevel caption: "LEVEL: 7"	Test 1: As expected Test 2: As expected	7 (Test 1 result is identical to test 2 result, therefore only one screenshot is displayed)
		8	Snake length: 72 bits long Snake facing: right Timer interval: 30 milliseconds lblLevel caption: "LEVEL: 8"	Test 1: As expected Test 2: As expected	8 (Test 1 result is identical to test 2 result, therefore only one screenshot is displayed)
		9	Snake length: 78 bits long Snake facing: right Timer interval: 20 milliseconds lblLevel caption: "LEVEL: 9"	Test 1: As expected Test 2: As expected	9 (Test 1 result is identical to test 2 result, therefore only one screenshot is displayed)
		10	Snake length: 84 bits long Snake facing: right Timer interval: 10 milliseconds lblLevel caption: "LEVEL: 10"	Test 1: As expected Test 2: As expected	10 (Test 1 result is identical to test 2 result, therefore only one screenshot is displayed)
Movement of snake	When in a game, pressing a directional button on the keyboard	Up (when head is facing right or left)	Snake's head will face and therefore move upwards, and each trailing snake bit will move to follow the path of the head every time the timer is called	As expected	11, 12
		Right (when head is facing up or down)	Snake's head will face and therefore move upwards, and each trailing snake bit will move to follow the path of the head every time the timer is called	As expected	13, 14
		Down (when	Snake's head will	As expected	15, 16

		head is facing right or left)	face and therefore move down, and each trailing snake bit will move to follow the path of the head every time the timer is called		
		Left (when head is facing up or down)	Snake's head will face and therefore move left, and each trailing snake bit will move to follow the path of the head every time the timer is called	As expected	17, 18
		Up (when head is facing up or down)	No change	As expected	19 (the results for when the head is facing up and down are identical, therefore only one screenshot is displayed)
		Right (when head is facing right or left)	No change	As expected	20 (the results for when the head is facing right and left are identical, therefore only one screenshot is displayed)
		Down (when head is facing up or down)	No change	As expected	21 (the results for when the head is facing right and left are identical, therefore only one screenshot is displayed)
		Left (when head is facing right or left)	No change	As expected	22 (the results for when the head is facing right and left are identical, therefore only one screenshot is

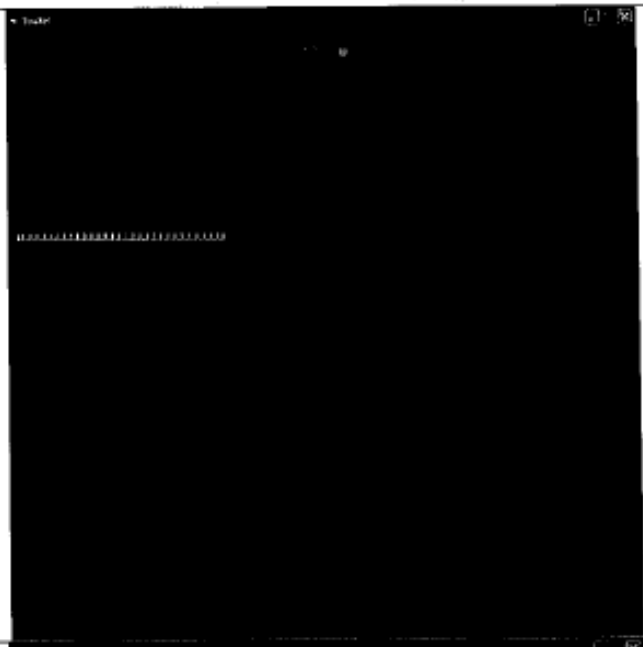
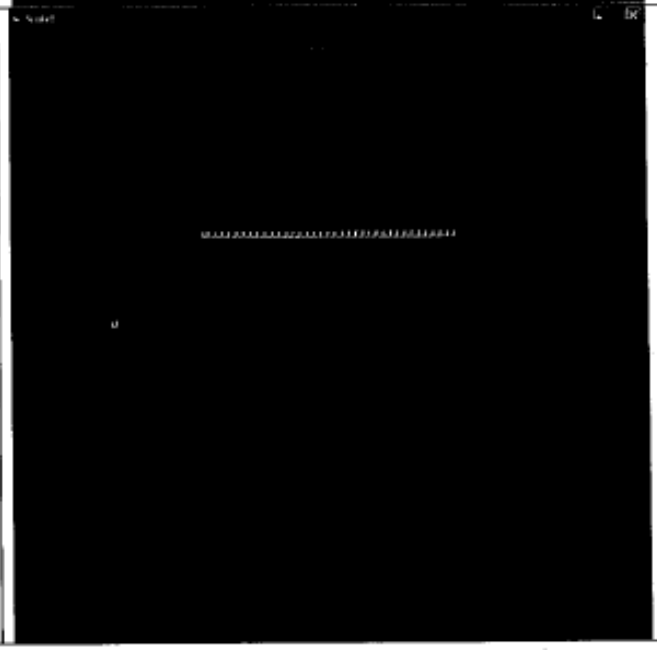
					displayed)
		None	Coordinates of snake's head are continually incremented by 10 pixels in the direction that the snake's head is facing and each trailing snake bit moves to follow the path of the head every time the timer is called	As expected	23-27 show bits 1 to 4 of the snake's body moving in the path of the head. 28 shows the head moving after all trailing bits have moved
Collision with walls	When in a game, directing snake's head into a border of the game screen using directional buttons on the keyboard	Up, to direct snake's head into top border of game screen	The instant the snake's head goes beyond the border of the game screen, an error message is produced and the game ends, returning to the initial menu screen	As expected	29 shows snake on the edge of the screen, about to collide. 30 shows the snake just beyond the screen, and the end game error message being displayed. 31 shows the initial menu screen that is returned to. This screen is the same as that seen in the following wall collision tests, so I will only display it once.
		Right, to direct snake's head into right border of game screen	As above	As expected	32 shows snake on the edge of the screen, about to collide. 33 shows the snake just beyond the screen, and the end game error message being displayed.

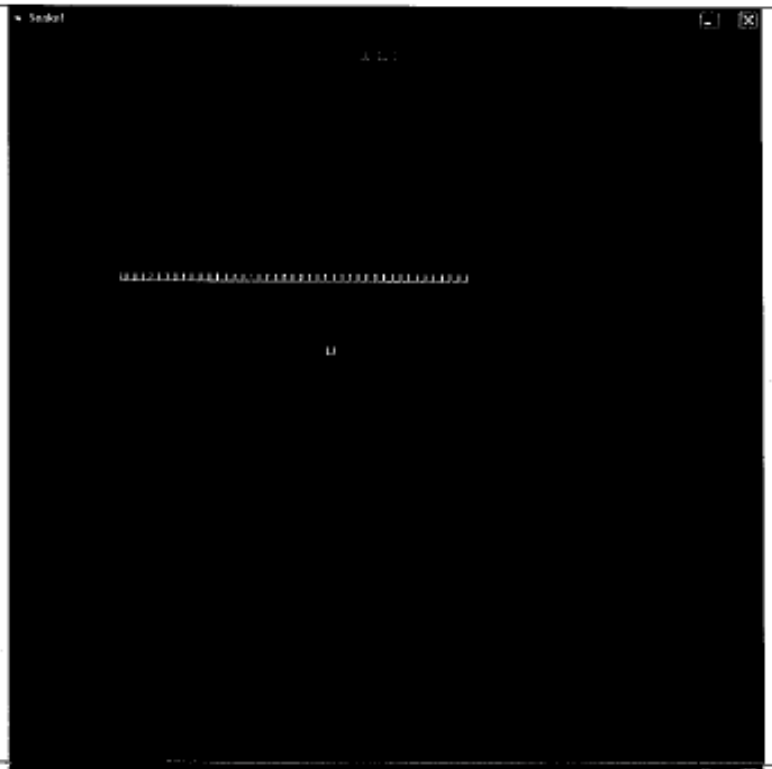
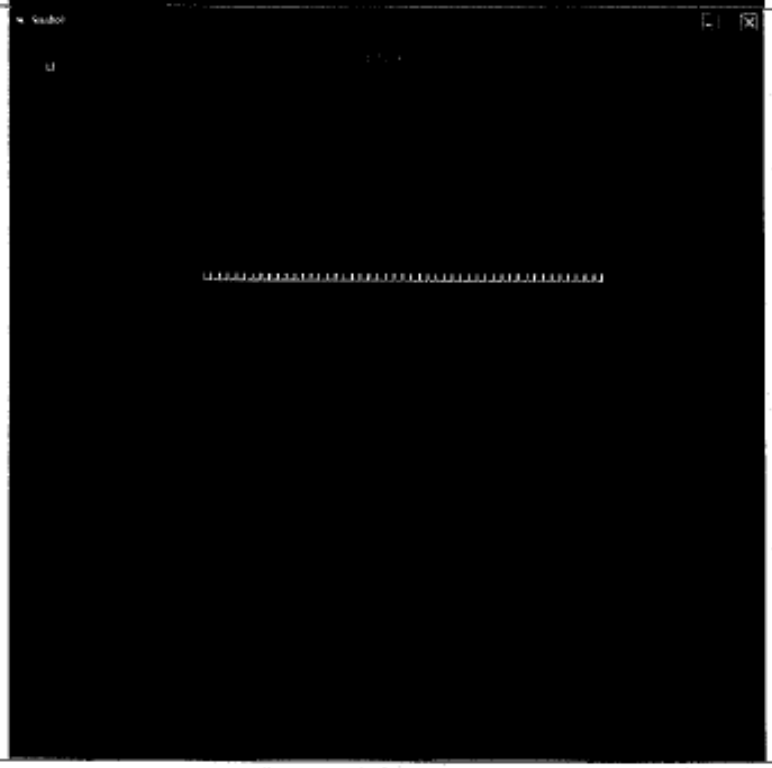
		Down, to direct snake's head into bottom border of game screen	As above	As expected	34 shows snake on the edge of the screen, about to collide. 35 shows the snake just beyond the screen, and the end game error message being displayed.
		Left, to direct snake's head into left border of game screen	As above	As expected	36 shows snake on the edge of the screen, about to collide. 37 shows the snake just beyond the screen, and the end game error message being displayed.
Collision with own body	When in a game, directing snake's head into its own body using directional buttons on the keyboard	Directional buttons on keyboard used to direct snake's head	The instant the snake's head has the same coordinates as any one of its trailing snake bits, an error message is produced and the game ends, returning to the initial menu screen	As expected	38 shows the snake the instant before its head collides with its body 39 shows the error message being displayed
Collision with food (with no level advancement)	When in a game, directing snake's head into a food item using directional buttons on the keyboard	Directional buttons on keyboard used to direct snake's head into a food item (excluding every fifth food item eaten)	Food item randomly moved to another position on the game screen (that is not currently occupied by a snake bit)	As expected – this test was carried out ten times to demonstrate that food was never placed on top of a snake bit, however only two screenshots are displayed as the results were similar each time	40 shows the snake's head just about to collide with food. 41 shows the food being moved to another position on the screen

Collision with food (with level advancement)	As above	Snake directed onto a food item when foodCount = 4, and level number is <= 10	Food item randomly moved to another position on the game screen (that is not currently occupied by a snake bit); Snake length: increased by 6 bits; Timer interval controlling snake speed: decreased by 10 milliseconds; lblLevel caption: "LEVEL: [level]"	As expected	42 shows the snake's head just about to collide with food. 43 shows the snake's length being increased, food being moved and lblLevel's caption being changed. 44 and 45 show the timer interval before and after it is altered between level 1 and 2
		Snake directed onto a food item when foodCount = 4, and level number is > 10	Food item randomly moved to another position on the game screen (that is not currently occupied by a snake bit); Snake length: increased by 6 bits; lblLevel caption: "LEVEL: [level]" No change to timer interval	As expected – No additional screenshots are shown because 42 and 43 demonstrate all that occurs	42, 43
Unlimited level advancement	Disable collision detection with own body or borders of game screen, then continually advance through levels up to level 50	NA	Up to and including level 10, the interval of the timer controlling the snake's speed will be decreased by 10 at every new level and snake length will be increased by 6 bits; Beyond level 10, the timer interval will remain at 10 milliseconds, and at every new level the snake length will be increased by 6 bits	Not quite as expected – the timer intervals and snake lengths were updated correctly, but beyond level 15 the snake began to slow down with each level completed. Because the timer intervals were correct, I attribute this to inefficient code controlling snake	46 shows the game at level 50. By this point the snake was moving much slower than it should have been.

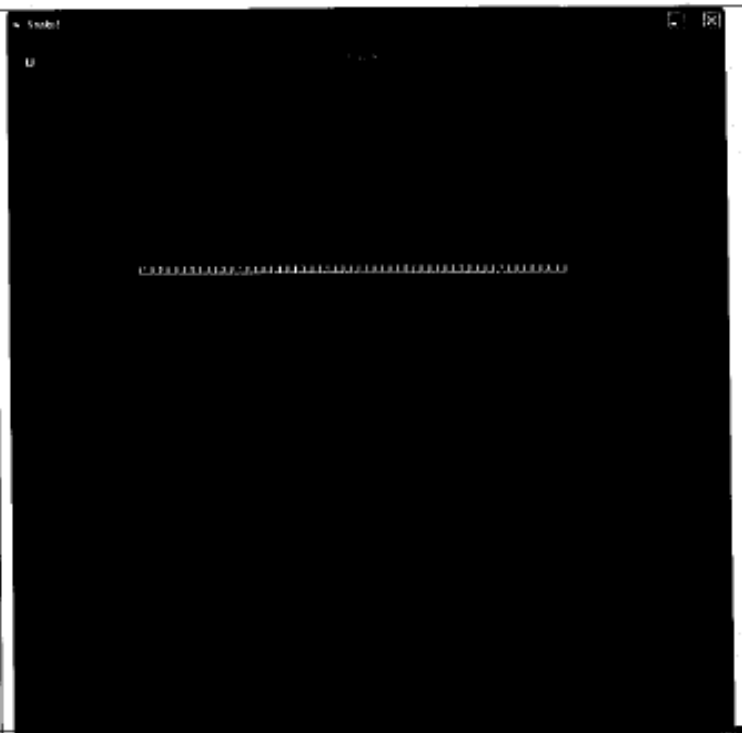
				movement,	
Displaying instructions	Clicking the instructions button (cmdInstructions) on the menu screen	NA	Displays instructions screen by setting the visible property of frmInstructions to True and the visible property of frmSnake to False	As expected	47 shows the instructions screen displayed after cmd-Instructions is clicked
Returning to initial menu screen from the instructions screen	Clicking the back button (cmdBack) on the instructions screen	NA	Returns to initial menu screen by setting the visible property of frmInstructions to False and the visible property of frmSnake to True	As expected	48 shows the initial menu screen displayed after cmdBack is clicked

Screenshots

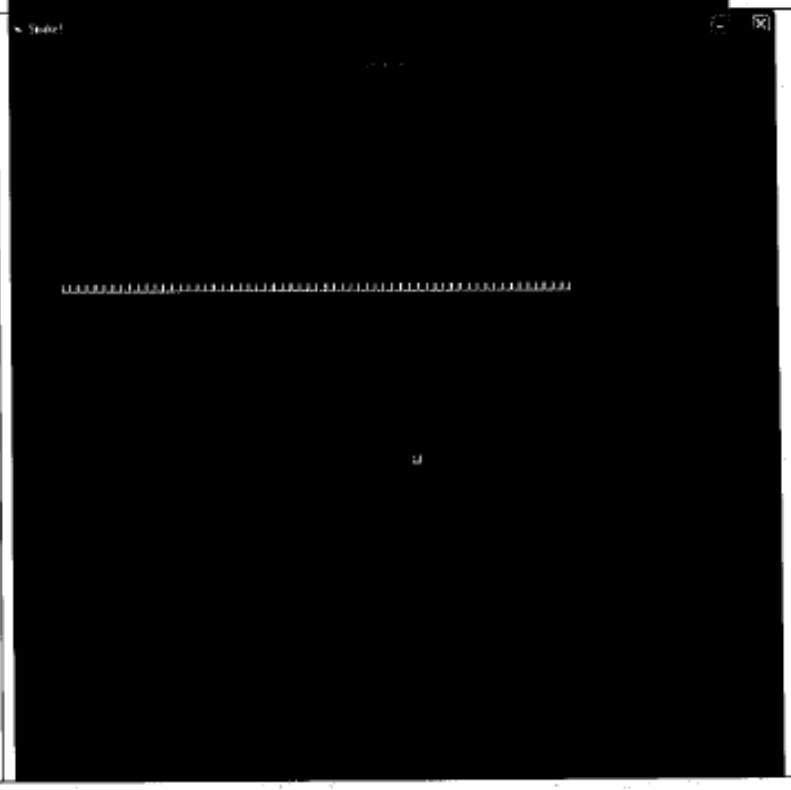
Screenshot reference	Screenshot
1	
2	

3	
4	

5

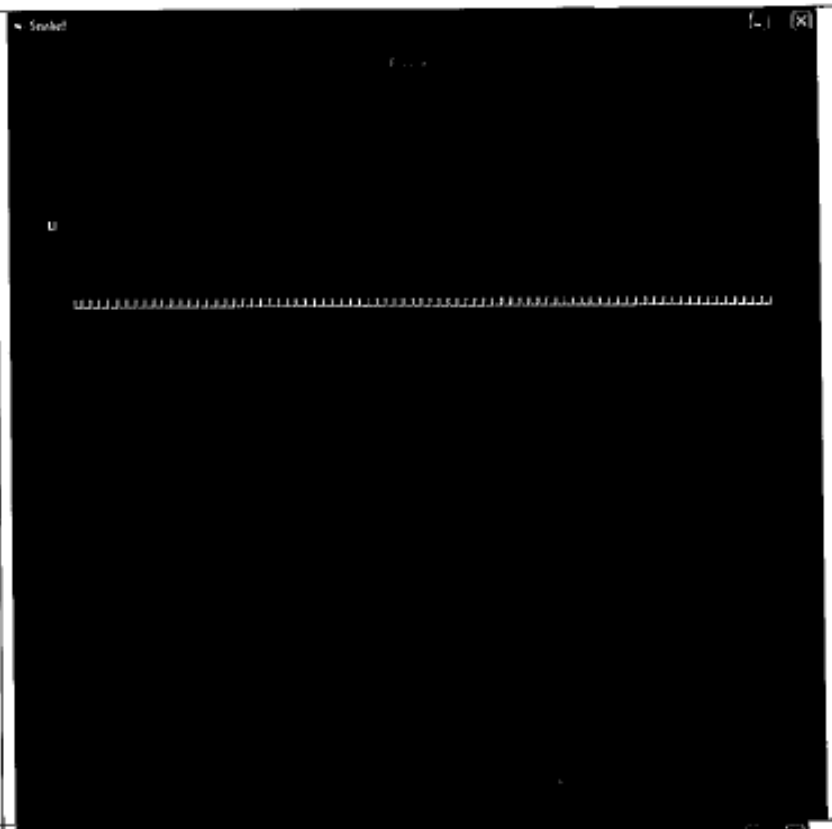


6

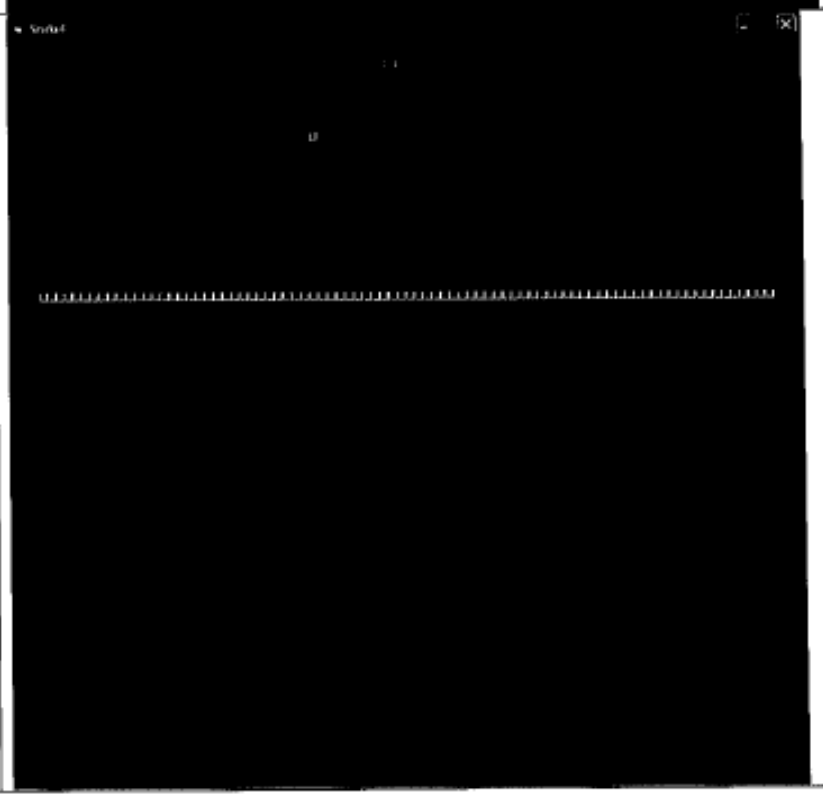


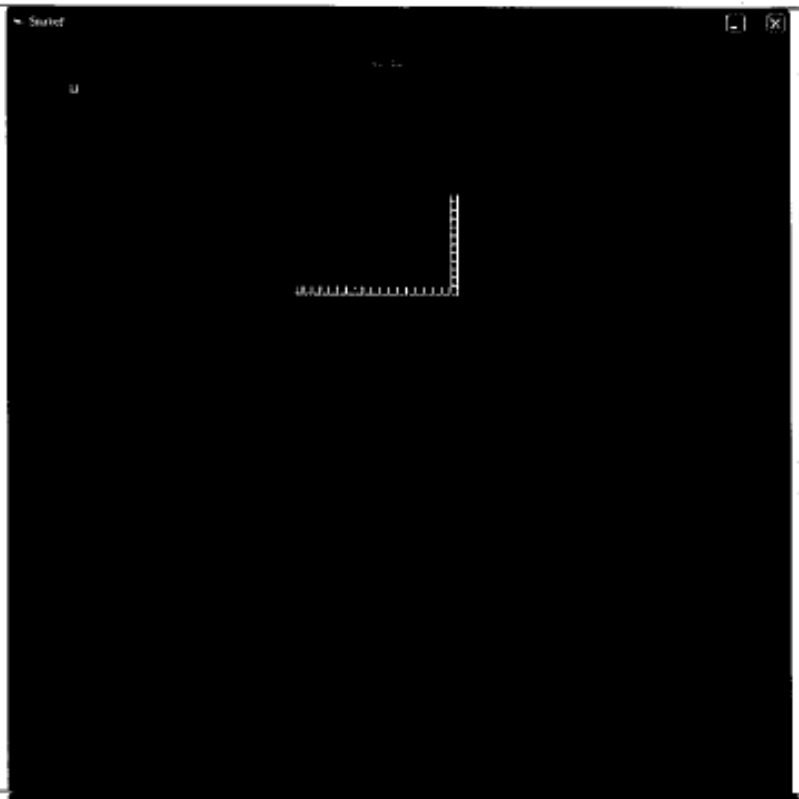
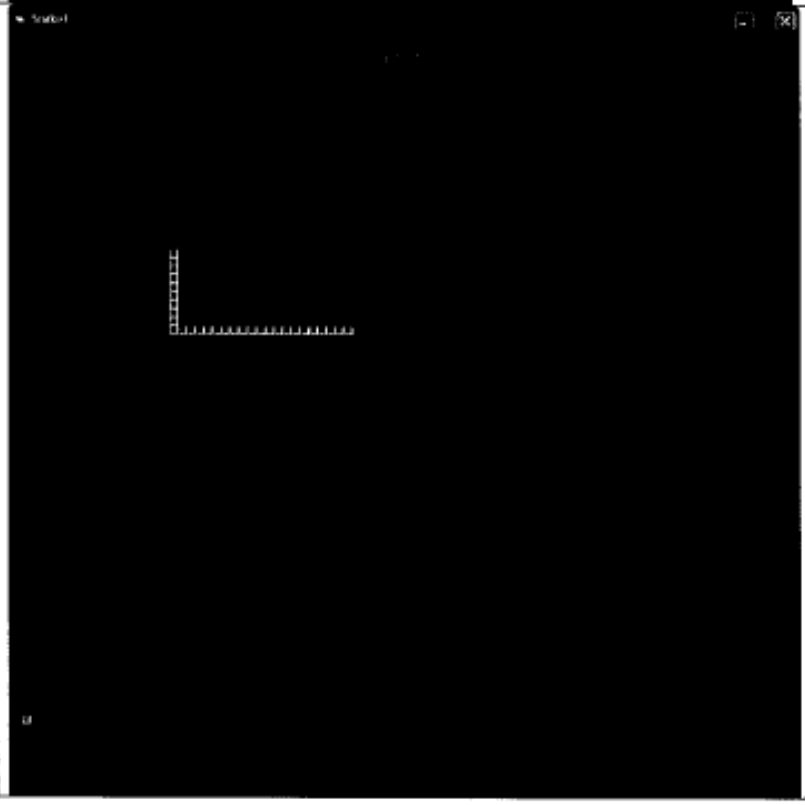
7	
8	

9



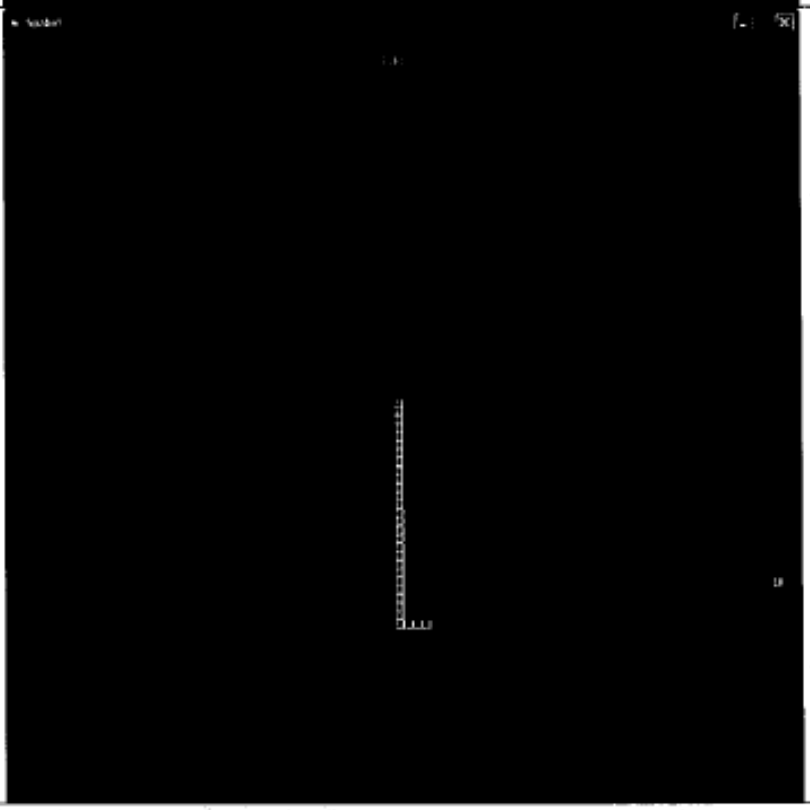
10



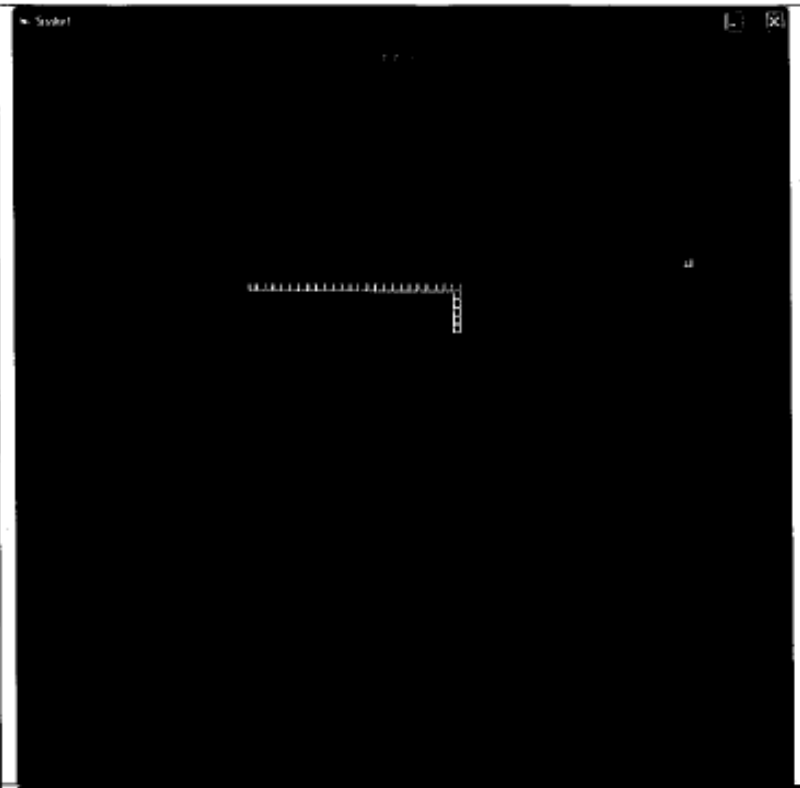
11	
12	

13

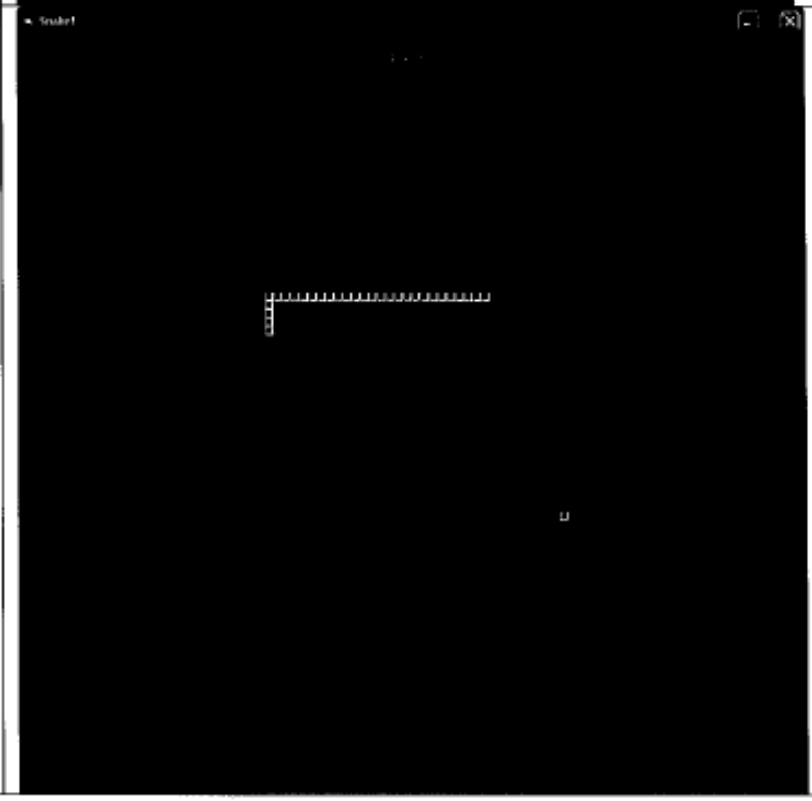
14



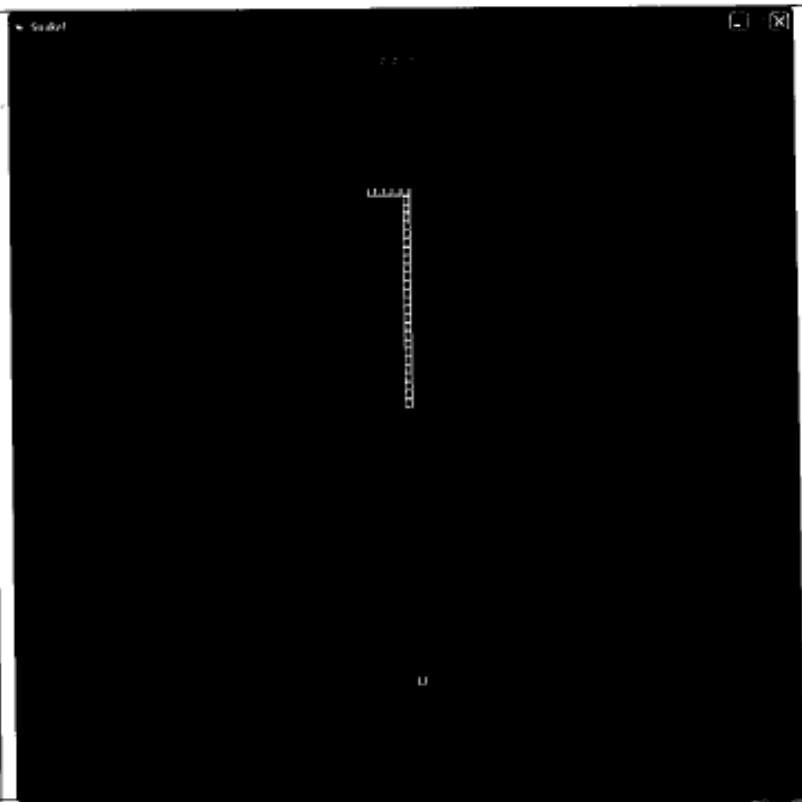
15



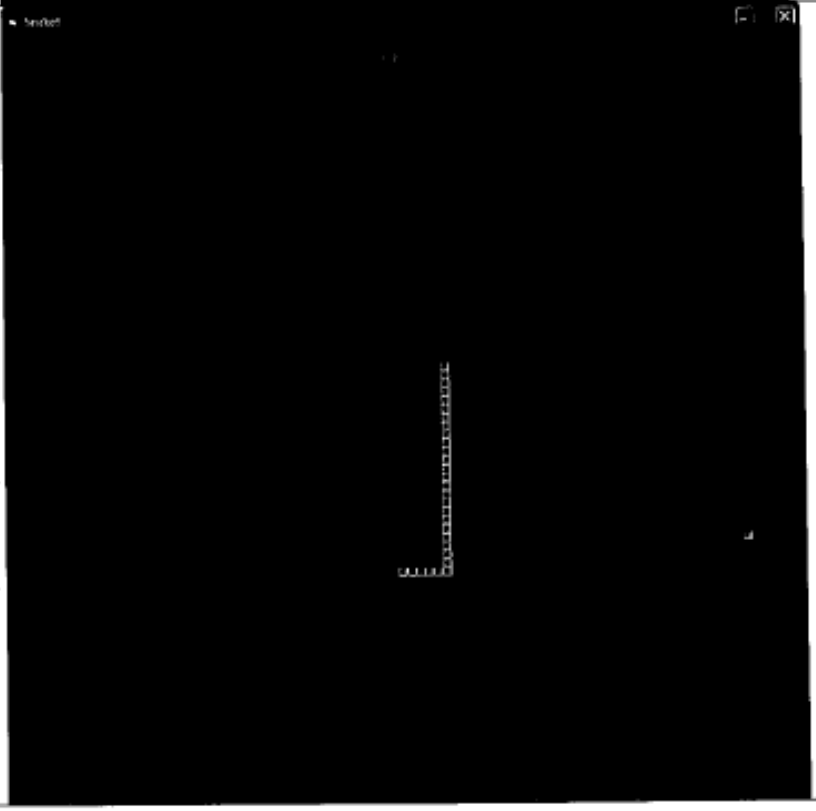
16

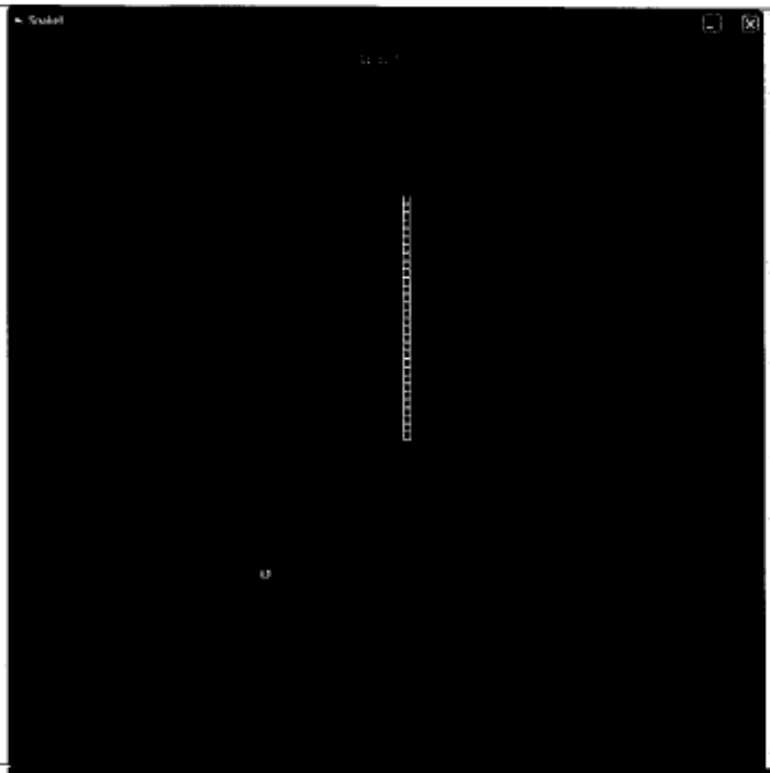
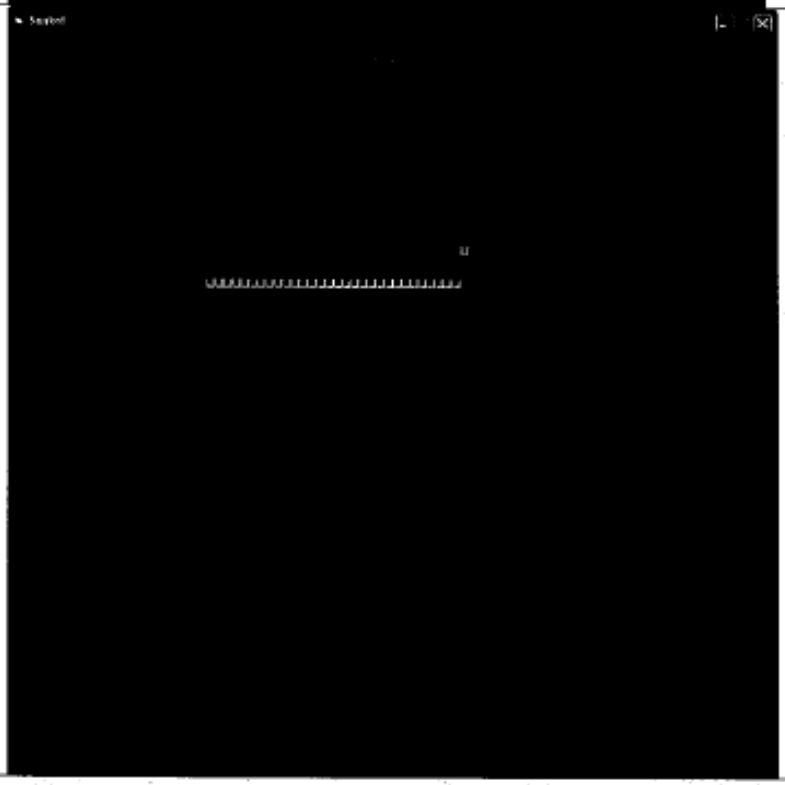


17

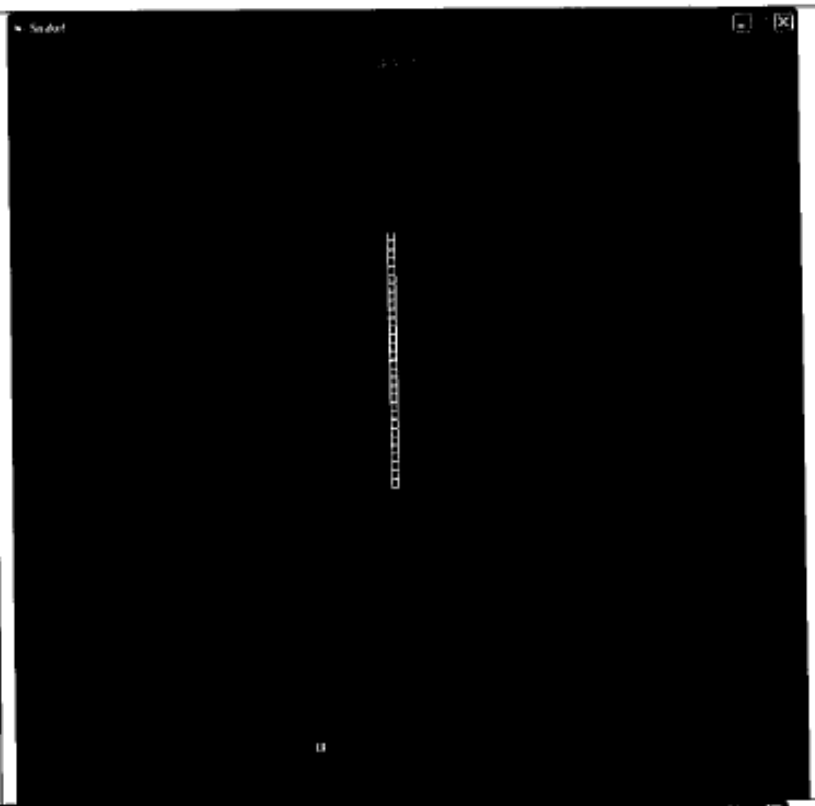


18

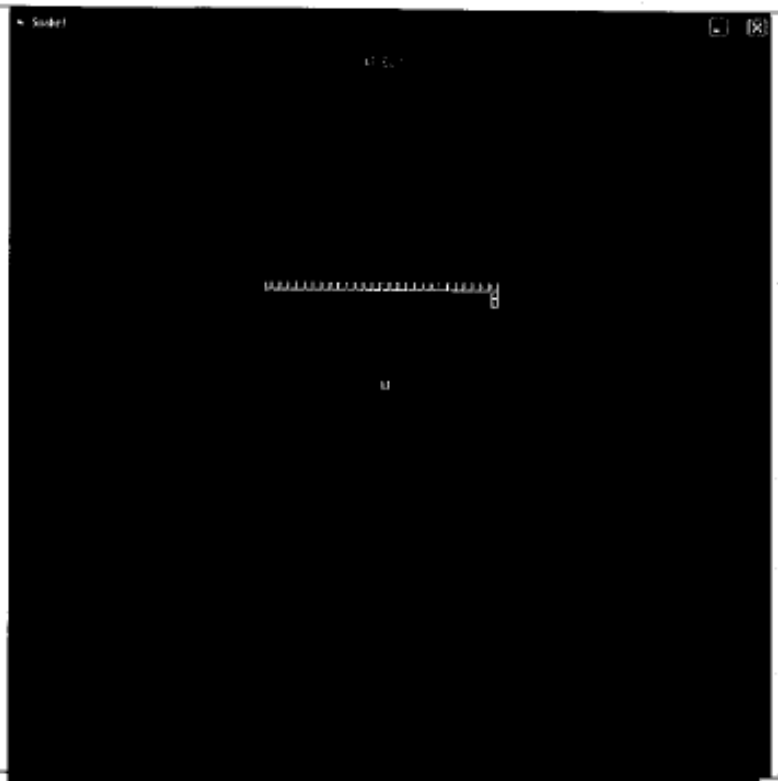
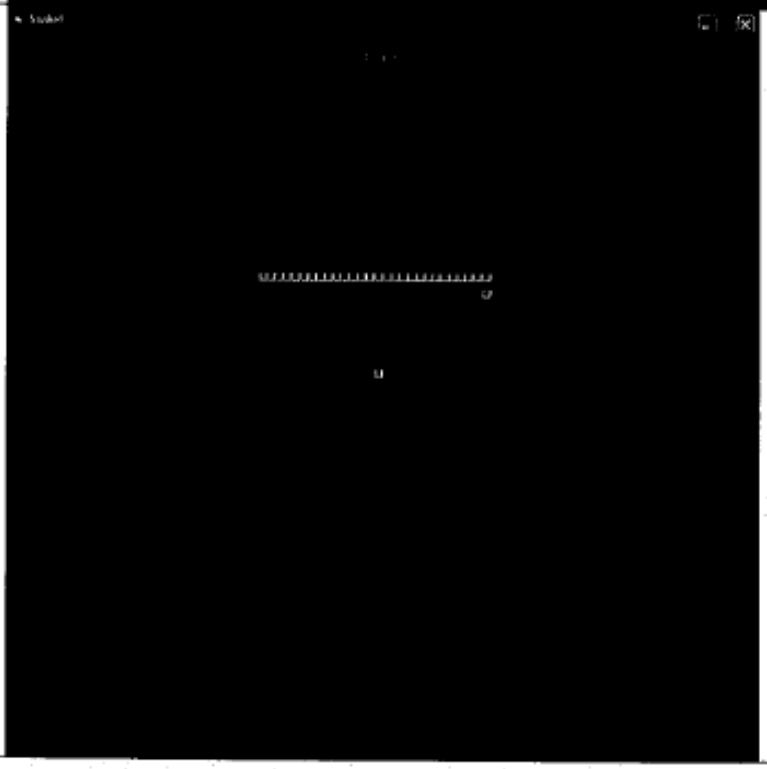


19	
20	

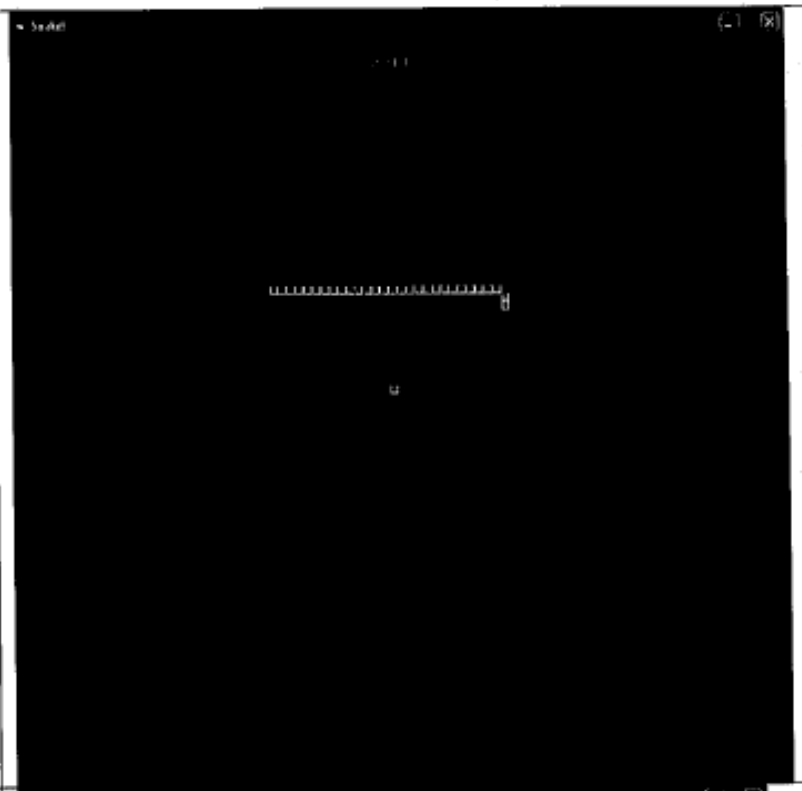
21



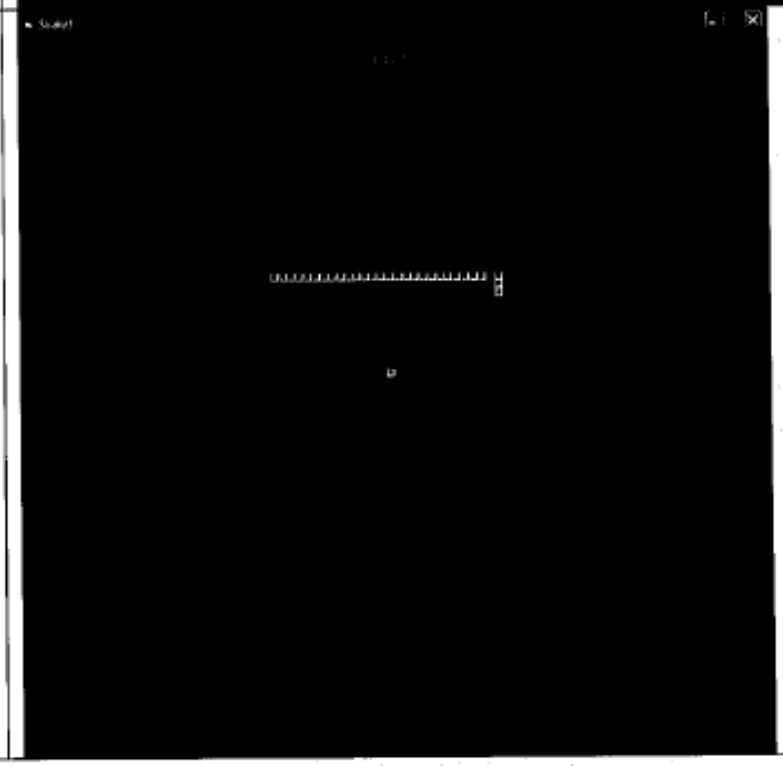
22

23	
24	

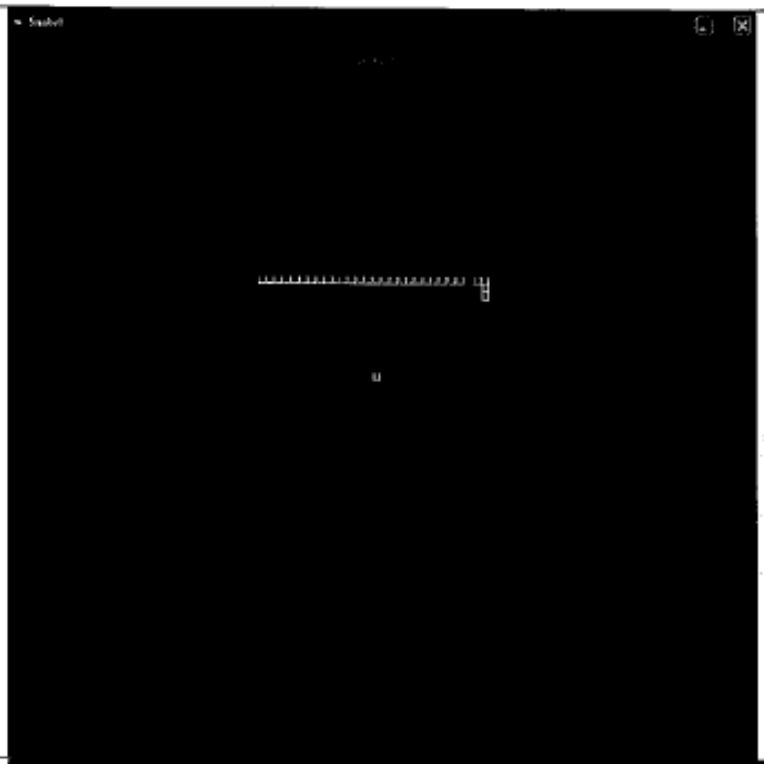
25



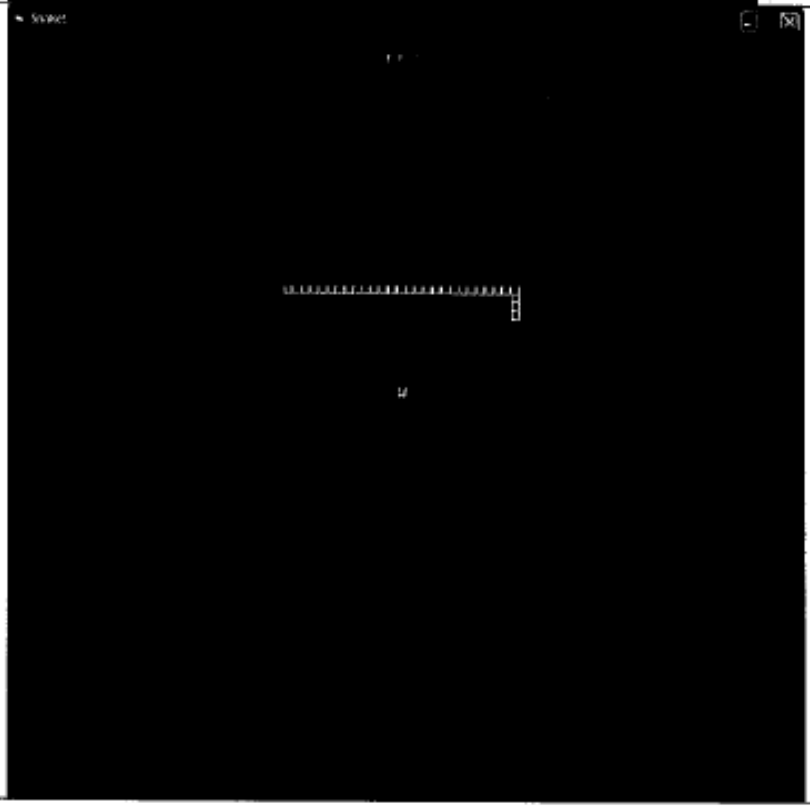
26



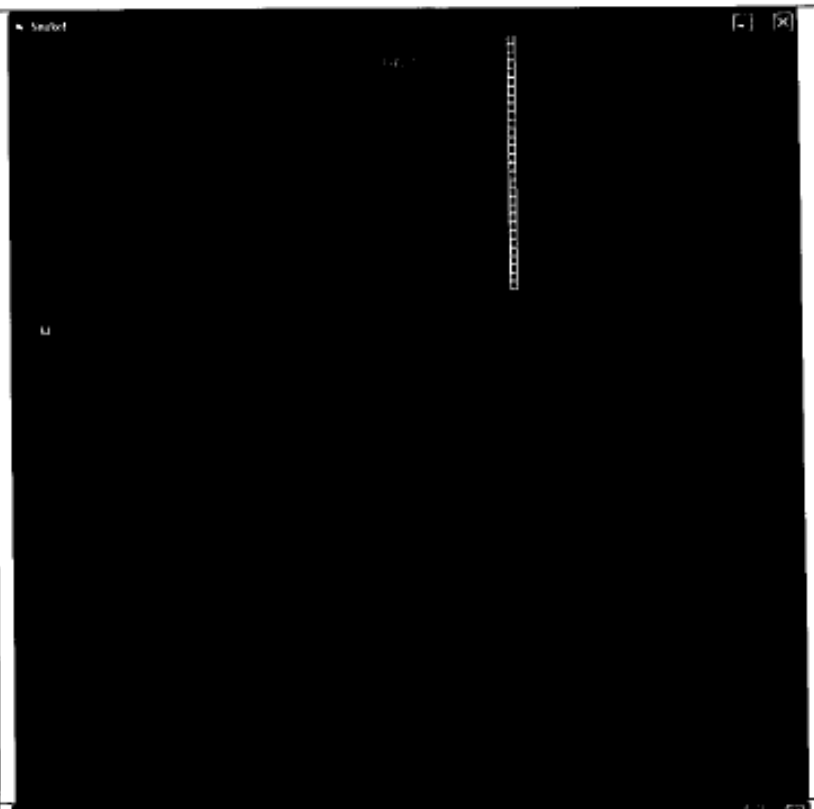
27



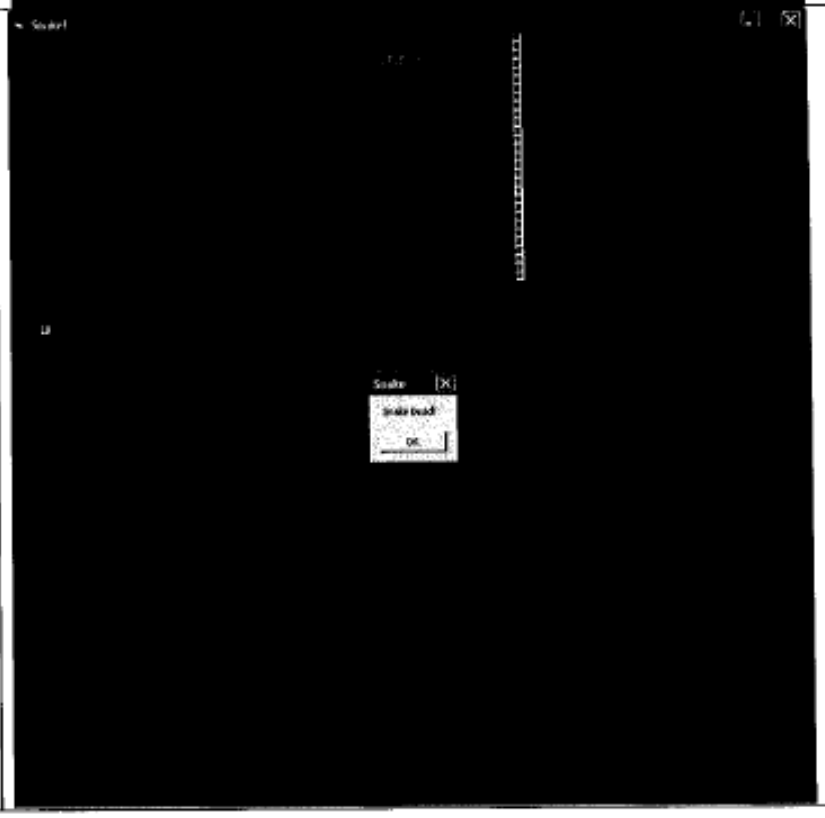
28



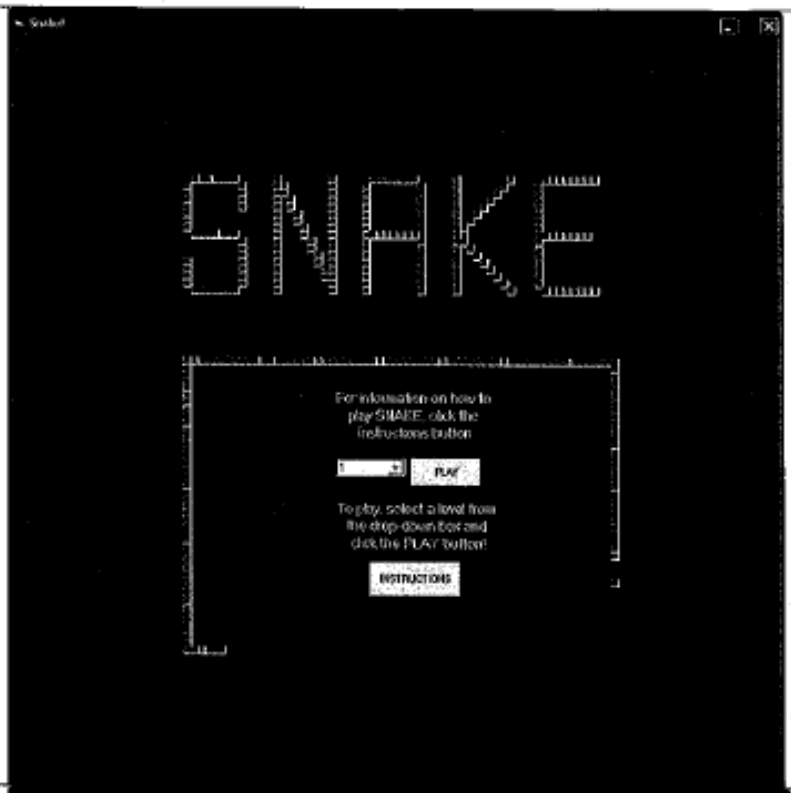
29



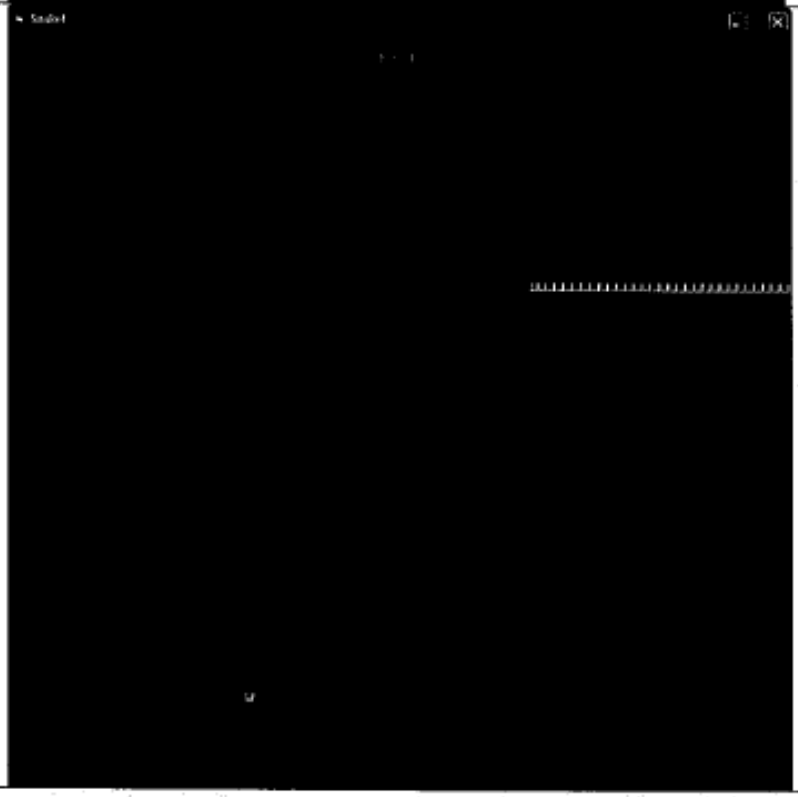
30

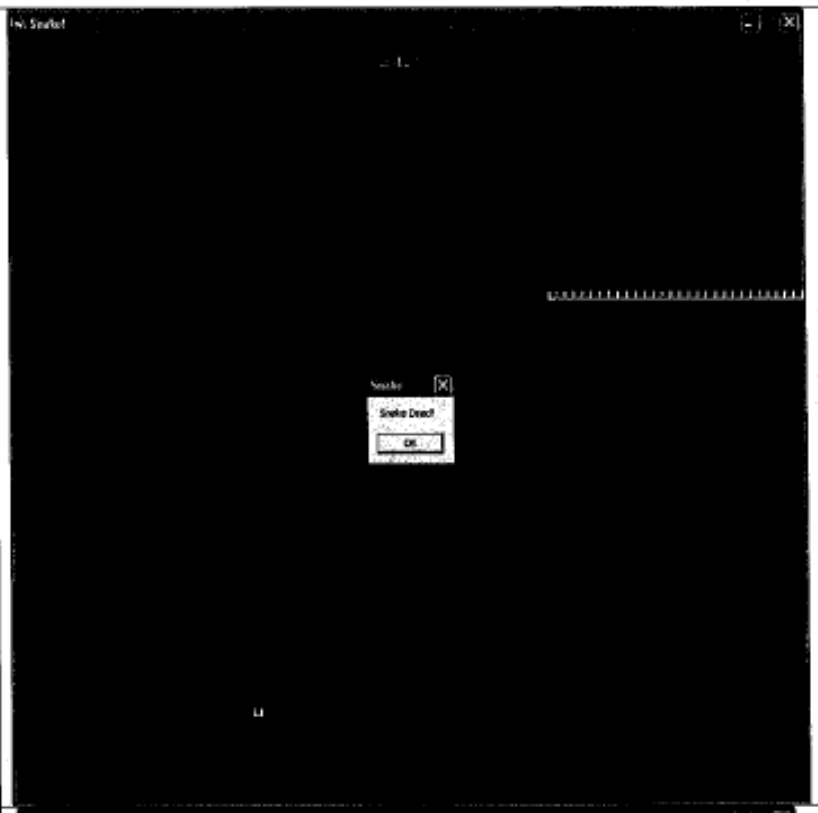
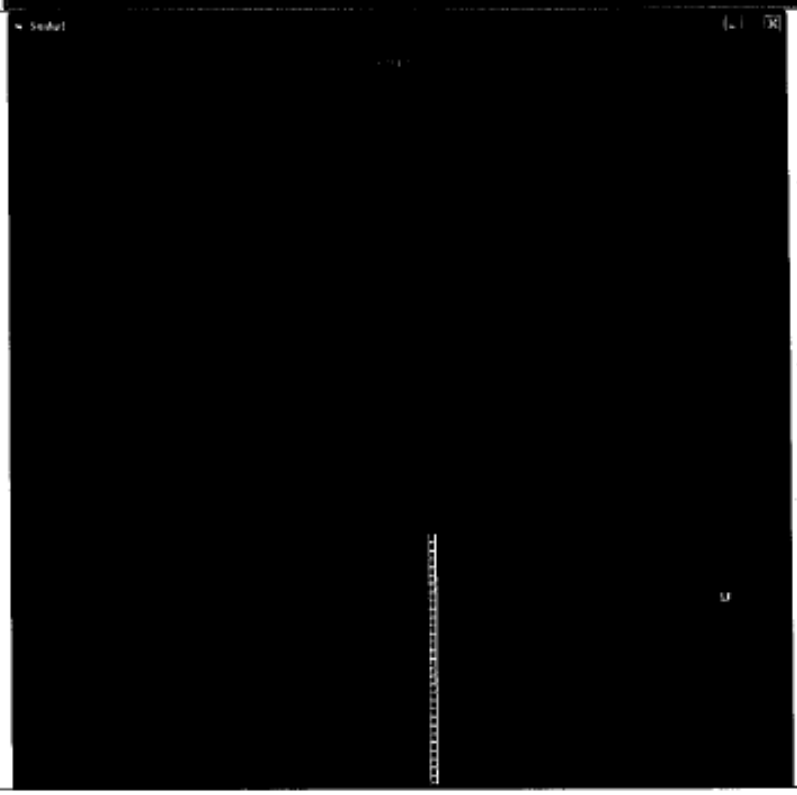


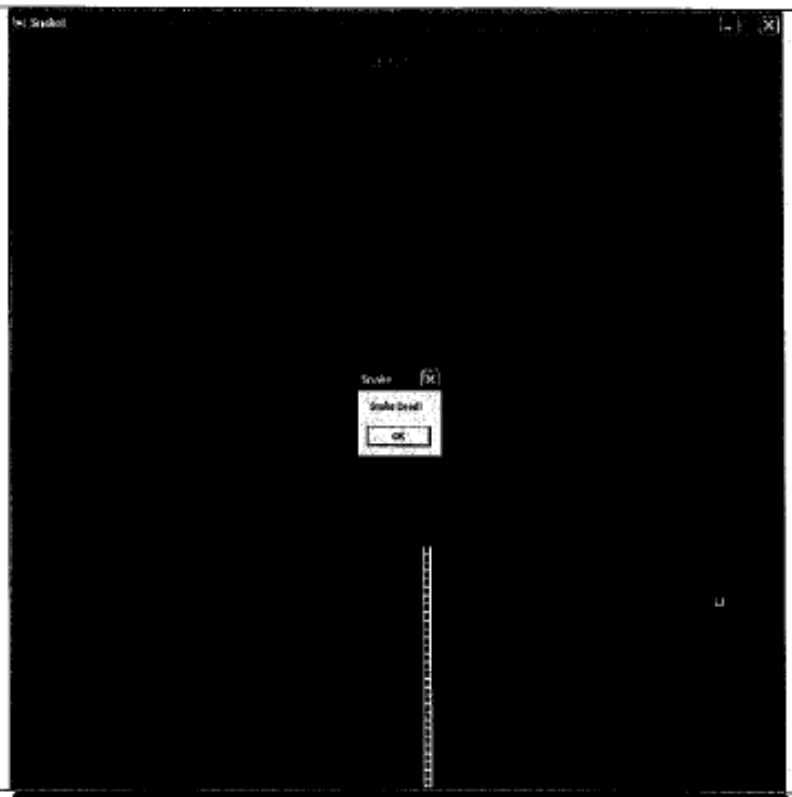
31


SNAKE
For information on how to play SNAKE, click the Instructions button
PLAY
To play, select a level from the drop-down box and click the PLAY button!
INSTRUCTIONS

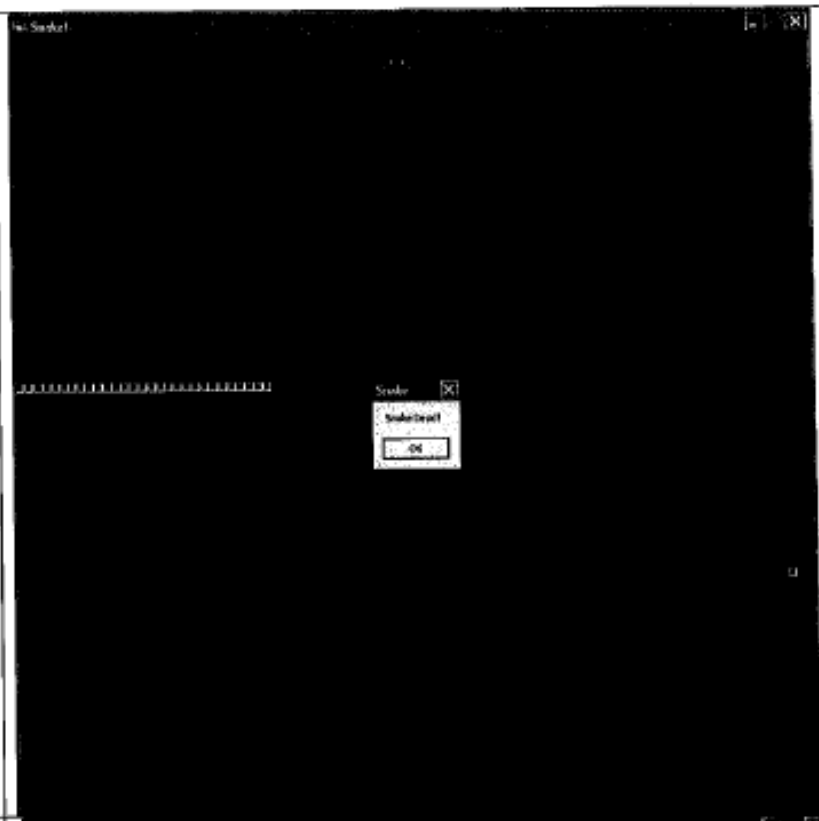
32



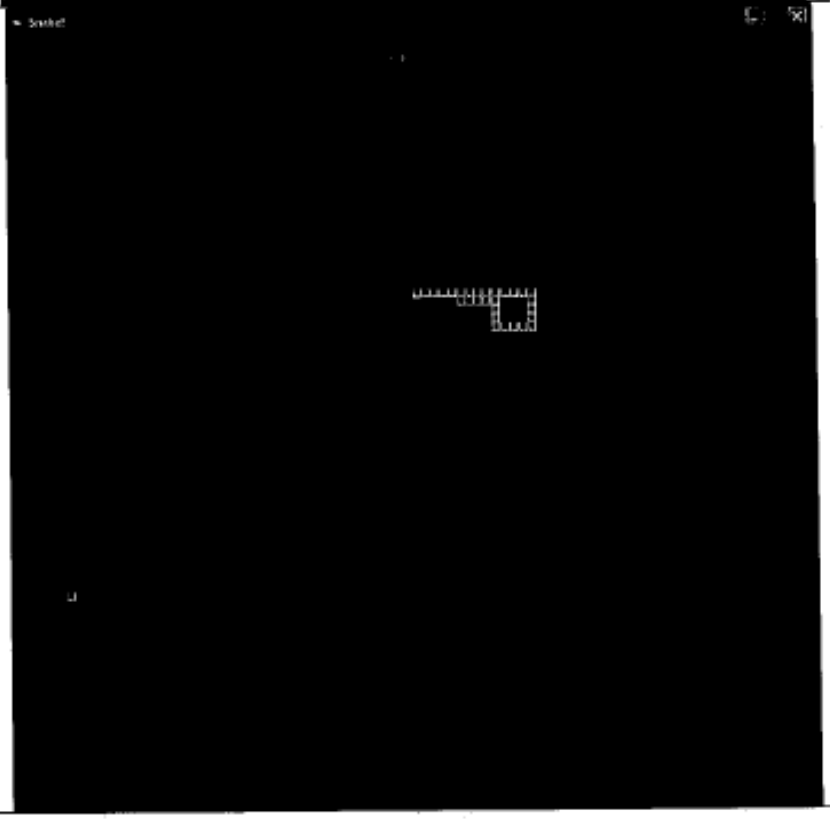
33	
34	

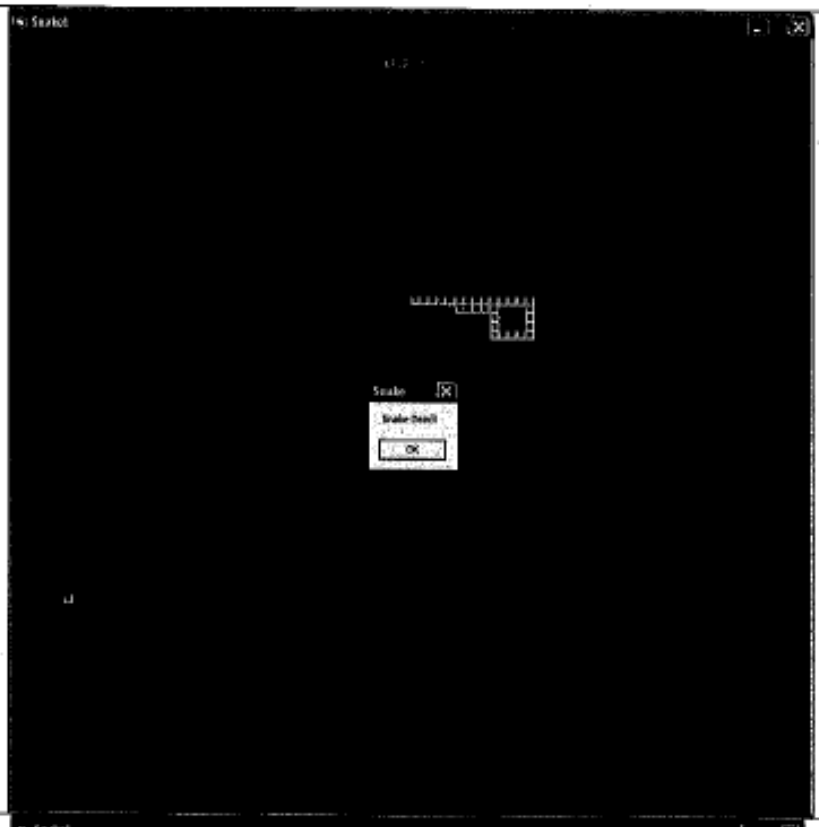
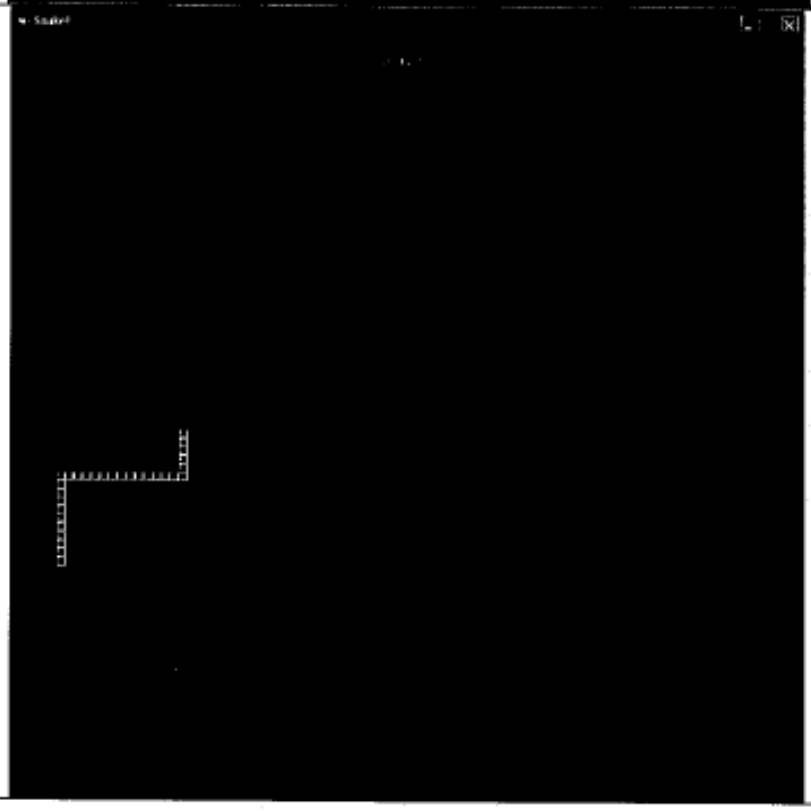
35	
36	

37

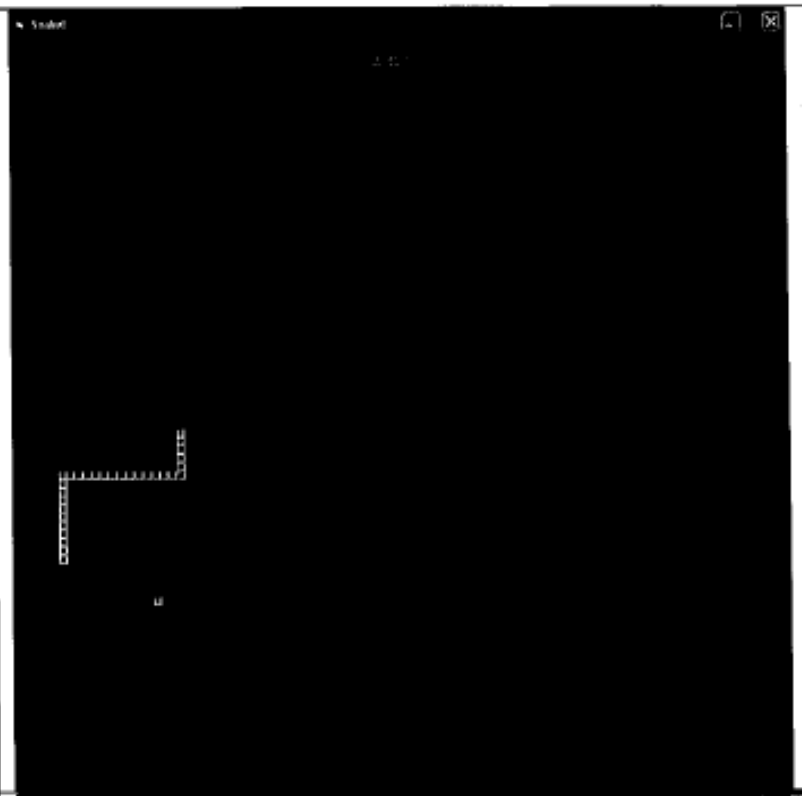


38

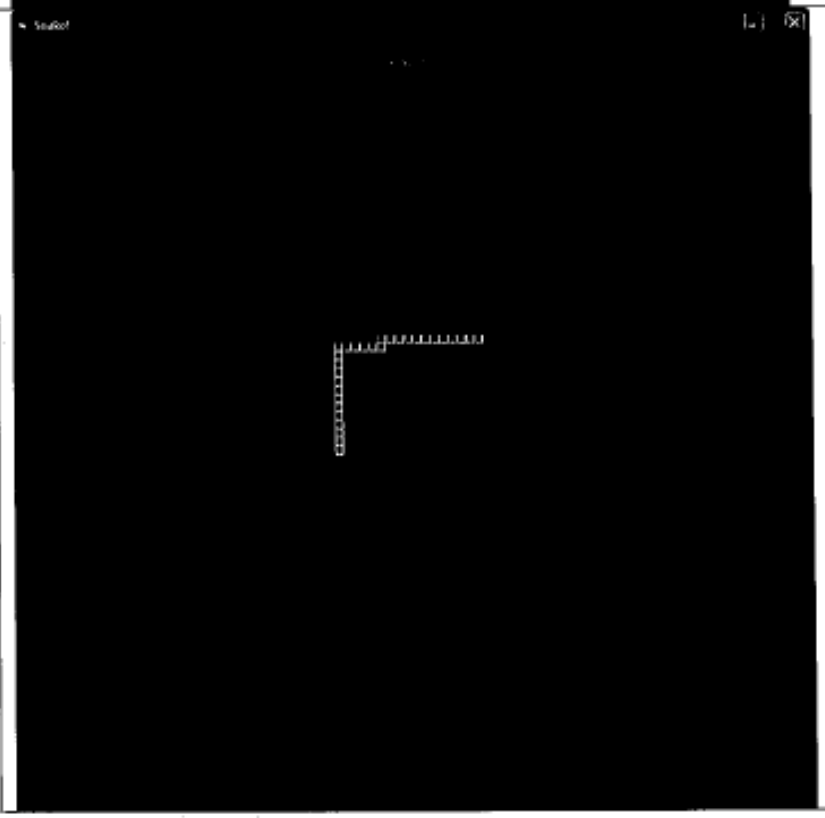


39	
40	

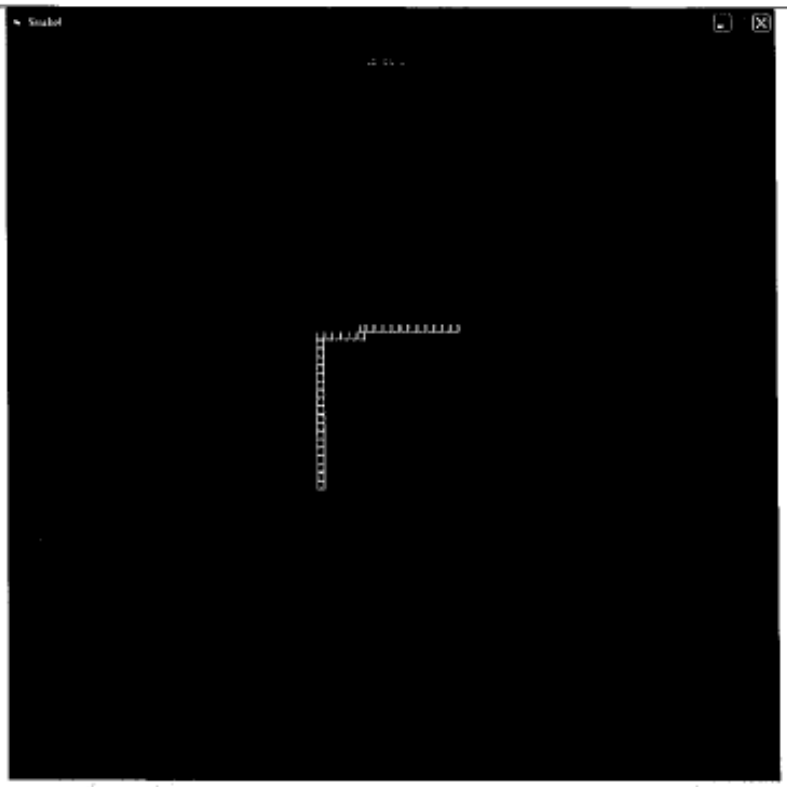
41



42



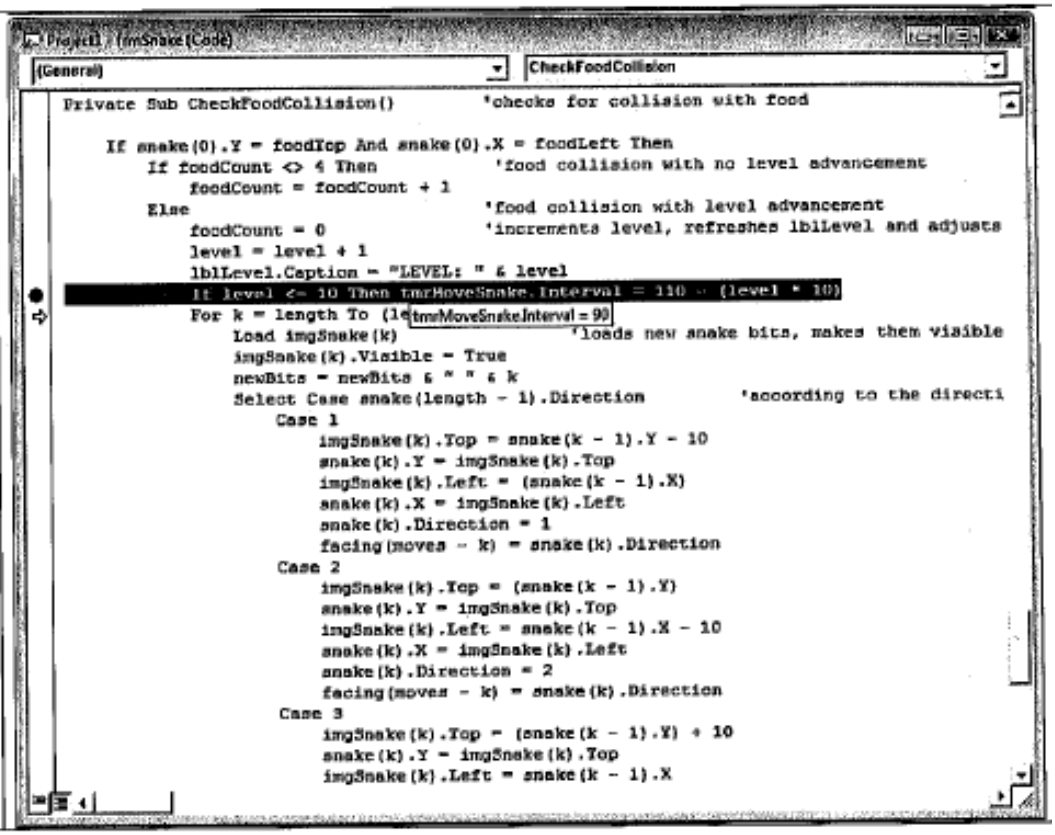
43



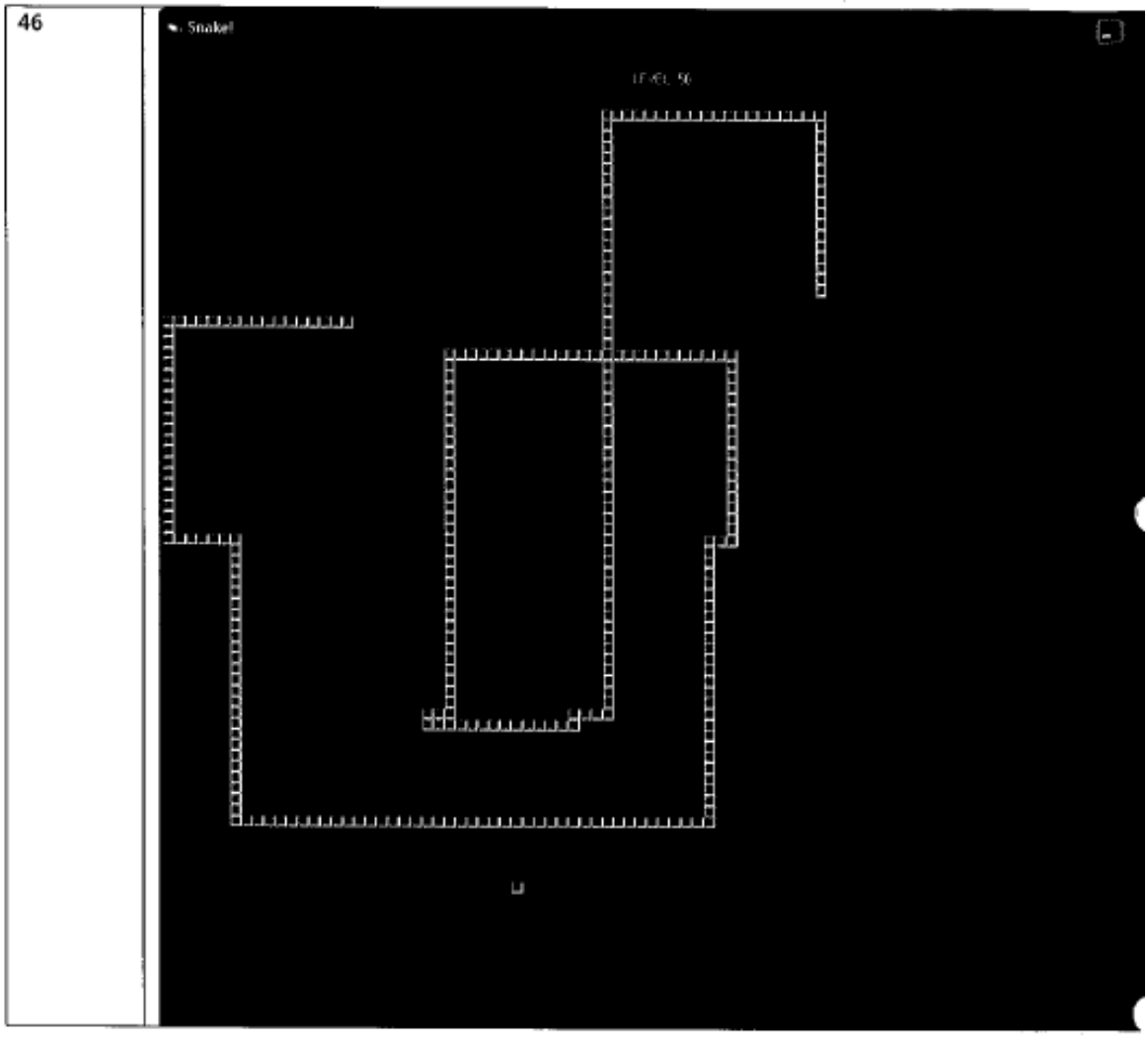
44

```
(General)                                   CheckFoodCollision

Private Sub CheckFoodCollision()        'checks for collision with food

    If snake(0).Y = foodTop And snake(0).X = foodLeft Then
        If foodCount <> 4 Then           'food collision with no level advancement
            foodCount = foodCount + 1
        Else                             'food collision with level advancement
            foodCount = 0                'increments level, refreshes lblLevel and adjust
            level = level + 1
            lblLevel.Caption = "LEVEL: " & level
            If level <= 10 Then tmrMoveSnake.Interval = 110 - (level * 10)
            For k = length To (l  tmrMoveSnake.Interval = 100
                Load imgSnake(k)                'loads new snake bits, makes them visible
                imgSnake(k).Visible = True
                newBits = newBits & " " & k
                Select Case snake(length - 1).Direction         'according to the directi
                    Case 1
                        imgSnake(k).Top = snake(k - 1).Y - 10
                        snake(k).Y = imgSnake(k).Top
                        imgSnake(k).Left = (snake(k - 1).X)
                        snake(k).X = imgSnake(k).Left
                        snake(k).Direction = 1
                        facing(moves - k) = snake(k).Direction
                    Case 2
                        imgSnake(k).Top = (snake(k - 1).Y)
                        snake(k).Y = imgSnake(k).Top
                        imgSnake(k).Left = snake(k - 1).X - 10
                        snake(k).X = imgSnake(k).Left
                        snake(k).Direction = 2
                        facing(moves - k) = snake(k).Direction
                    Case 3
                        imgSnake(k).Top = (snake(k - 1).Y) + 10
                        snake(k).Y = imgSnake(k).Top
                        imgSnake(k).Left = snake(k - 1).X
```

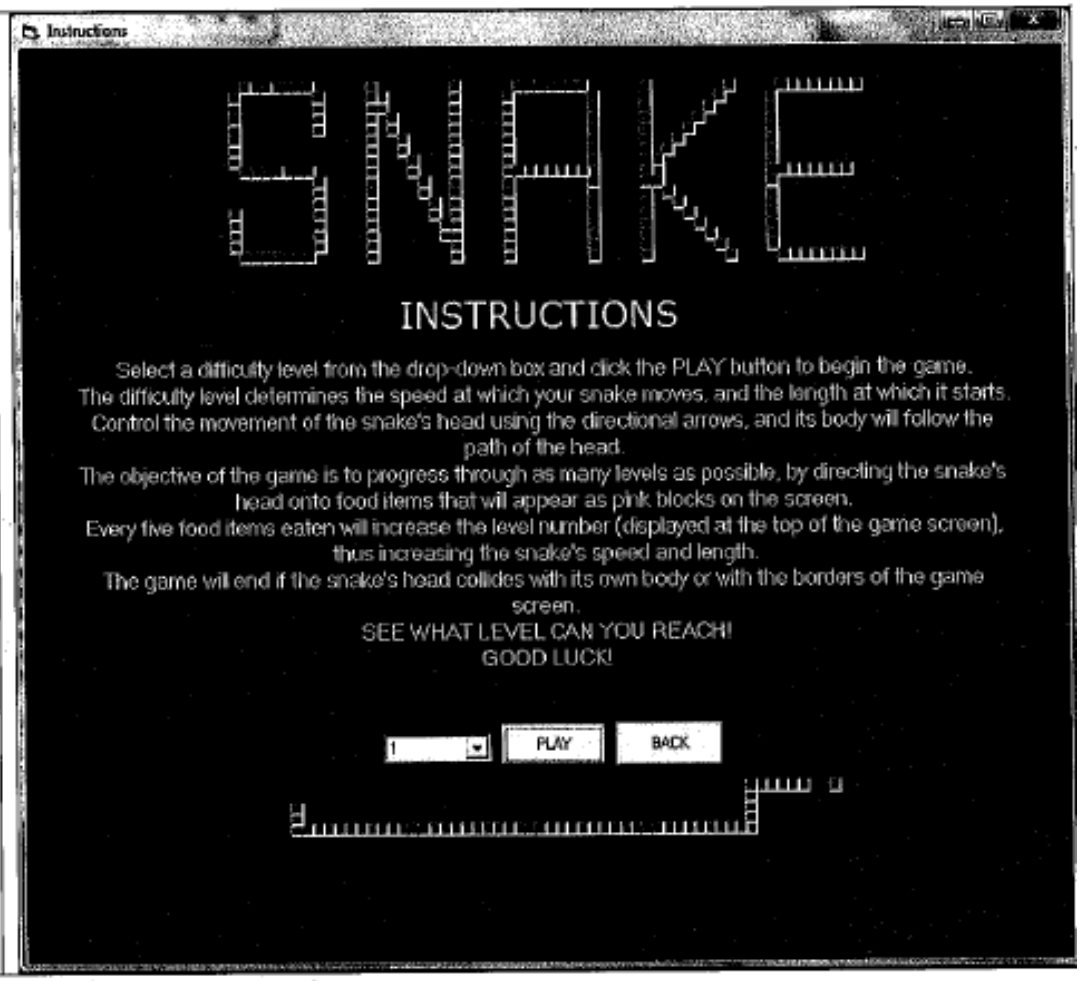
45


frmSnake (Code)
(General)
CheckFoodCollision
Private Sub CheckFoodCollision() 'checks for collision with food
If snake(0).Y = foodTop And snake(0).X = foodLeft Then
If foodCount <> 4 Then 'food collision with no level advancement
foodCount = foodCount + 1
Else 'food collision with level advancement
foodCount = 0 'increments level, refreshes lblLevel and adjusts
level = level + 1
lblLevel.Caption = "LEVEL: " & level
If level <= 10 Then tmrMoveSnake.Interval = 110 - (level * 10)
tmrMoveSnake.Interval = 90
For k = length To (le
Load imgSnake(k) 'loads new snake bits, makes them visible
imgSnake(k).Visible = True
newBits = newBits & " " & k
Select Case snake(length - 1).Direction 'according to the directi
Case 1
imgSnake(k).Top = snake(k - 1).Y - 10
snake(k).Y = imgSnake(k).Top
imgSnake(k).Left = (snake(k - 1).X)
snake(k).X = imgSnake(k).Left
snake(k).Direction = 1
facing(moves - k) = snake(k).Direction
Case 2
imgSnake(k).Top = (snake(k - 1).Y)
snake(k).Y = imgSnake(k).Top
imgSnake(k).Left = snake(k - 1).X - 10
snake(k).X = imgSnake(k).Left
snake(k).Direction = 2
facing(moves - k) = snake(k).Direction
Case 3
imgSnake(k).Top = (snake(k - 1).Y) + 10
snake(k).Y = imgSnake(k).Top
imgSnake(k).Left = snake(k - 1).X

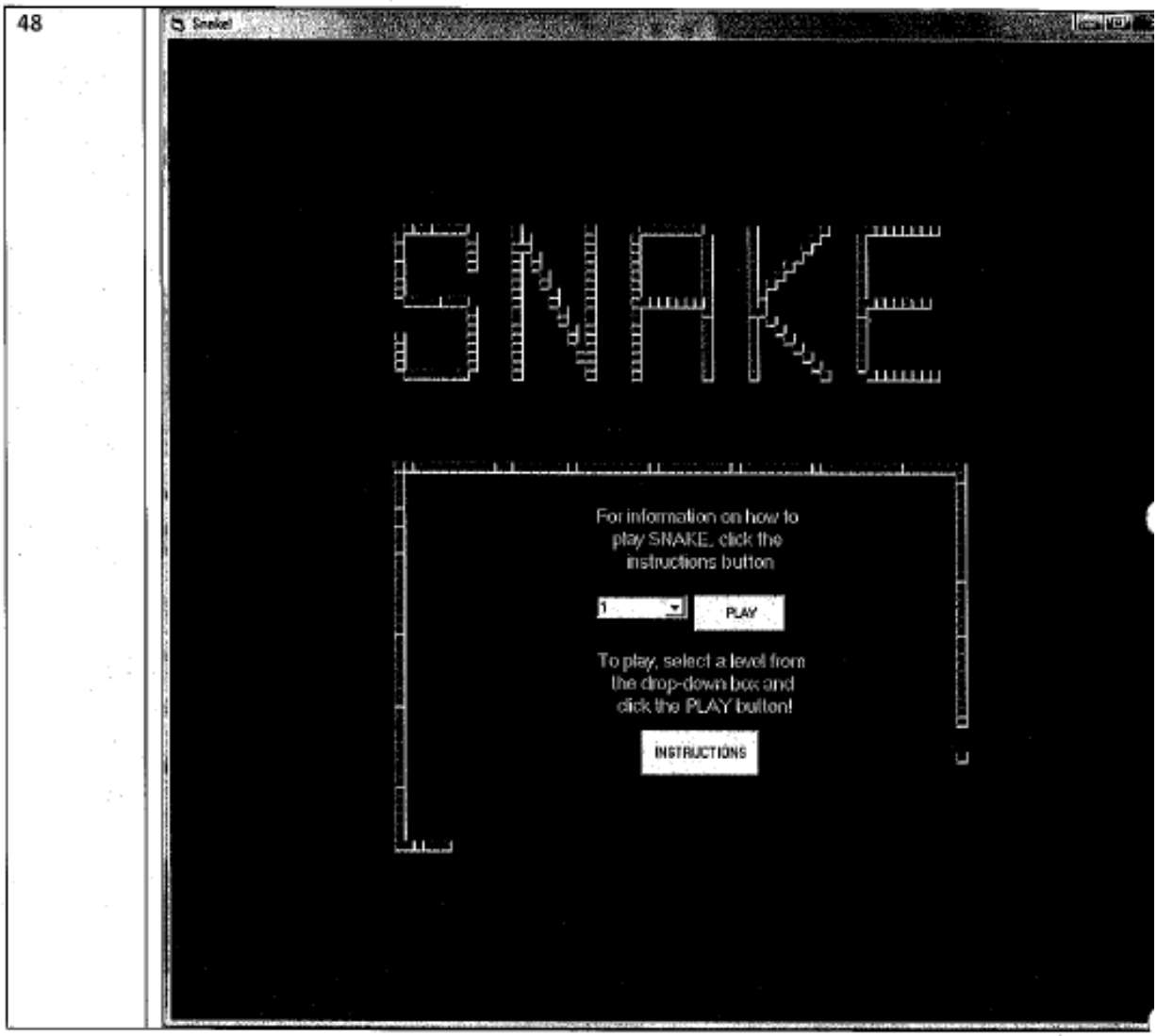
46



47


Instructions
SNAKE
INSTRUCTIONS
Select a difficulty level from the drop-down box and click the PLAY button to begin the game.
The difficulty level determines the speed at which your snake moves, and the length at which it starts.
Control the movement of the snake's head using the directional arrows, and its body will follow the path of the head.
The objective of the game is to progress through as many levels as possible, by directing the snake's head onto food items that will appear as pink blocks on the screen.
Every five food items eaten will increase the level number (displayed at the top of the game screen), thus increasing the snake's speed and length.
The game will end if the snake's head collides with its own body or with the borders of the game screen.
SEE WHAT LEVEL CAN YOU REACH!
GOOD LUCK!
1
PLAY
BACK

48


Snake!
SNAKE
For information on how to play SNAKE, click the instructions button
1
PLAY
To play, select a level from the drop-down box and click the PLAY button!
INSTRUCTIONS

Result of testing

By testing the program according to my test plan, I have shown that all features are functional, with one exception. The snake speed increases as desired with every level increase up to level 10, and remains constant, as desired, up to level 12. However, with level increases above level 12, the snake appears to slow down. As the timer interval is still correct at such levels (it has the desired value of 10 milliseconds), I believe that the cause of the slow-down is the result of the inefficiency of the code controlling the snake's movement.

I have partially fixed this problem by reducing the amount of for...next loops executed with each snake movement. I did this by incorporating the CheckSnakeCollision subroutine into the MoveSnakeBody subroutine, whilst swapping the i's and k's in the snake collision detection algorithm around. This is only a partial solution, as the snake still appears to slow down at levels above level 15. To fully fix the problem, the method of snake movement would have to be fundamentally changed, and so I shall leave this to be a consideration in the project's evaluation.

These changes are displayed below:

Original separate code:

MoveSnakeBody

```
For i = 1 To (length - 1)
  If InStr(newBits, " " & i & " ") = 0 Then
    snake(i).Direction = facing(moves - i)
  End If
  If i = length - 1 Then newBits = ""
  Select Case snake(i).Direction
    Case 1
    snake(i).Y = snake(i).Y + 10
    imgSnake(i).Top = snake(i).Y
    Case 2
    snake(i).X = snake(i).X + 10
    imgSnake(i).Left = snake(i).X
    Case 3
    snake(i).Y = snake(i).Y - 10
    imgSnake(i).Top = snake(i).Y
    Case 4
    snake(i).X = snake(i).X - 10
    imgSnake(i).Left = snake(i).X
  End Select
Next i
```

CheckSnakeCollision

```
For k = 1 To (length - 1)
  If snake(0).Y = snake(k).Y And snake(0).X = snake(k).X Then
    tmrMoveSnake.Enabled = False
    MsgBox ("Snake Dead!")
```

```
      For i = 0 To (length - 1)
        imgSnake(i).Visible = False
        If i > 4 Then Unload imgSnake(i)
      Next i
      Call EndGame
    End If
  Next k
```

Combined subroutine called CheckSnakeCollisionAndMoveSnakeBody:

```
  For i = 1 To (length - 1)
    If snake(0).Y = snake(i).Y And snake(0).X = snake(i).X Then
      tmrMoveSnake.Enabled = False
      MsgBox ("Snake Dead!")
      For k = 0 To (length - 1)
        imgSnake(k).Visible = False
        If k > 4 Then Unload imgSnake(k)
      Next k
      Call EndGame
      Exit Sub
    End If
    If InStr(newBits, " " & i & " ") = 0 Then
      snake(i).Direction = facing(moves - i)
    End If
    If i = length - 1 Then newBits = ""
    Select Case snake(i).Direction
      Case 1
      snake(i).Y = snake(i).Y + 10
      imgSnake(i).Top = snake(i).Y
      Case 2
      snake(i).X = snake(i).X + 10
      imgSnake(i).Left = snake(i).X
      Case 3
      snake(i).Y = snake(i).Y - 10
      imgSnake(i).Top = snake(i).Y
      Case 4
      snake(i).X = snake(i).X - 10
      imgSnake(i).Left = snake(i).X
    End Select
  Next i
```

Evidence of improvements

In order to evidence the above improvements I shall time how long it takes for the snake to move from the left to the right hand side of the game screen, therefore covering a distance of 900 pixels, and then I shall divide the result by 90 to give the amount of time it takes for the snake to make a single movement. I shall perform this test for levels 10 to 20 using both the old and the new code. The results are displayed in the table below.

Level	Time taken to move 900 pixels (milliseconds)		Time taken to move 10 pixels (milliseconds)	
	Original code	Altered code	Original code	Altered code
10	920	910	10.2	10.1
11	940	920	10.4	10.2
12	960	920	10.7	10.2
13	1310	940	14.5	10.4
14	1660	990	18.4	11
15	1990	1040	22.1	11.5
16	2300	1290	25.5	14.3
17	2720	1510	30.2	16.7
18	3020	1820	33.5	20.2
19	3660	2050	40.6	22.7
20	4120	2340	45.7	26

As can be seen, there is a marked improvement in the degree to which the snake slows down when the altered code is run. However, as previously mentioned, this is clearly only a partial fix – a new version of the program that could overcome this problem will be proposed in the evaluation stage of the project.

User testing

I have discussed with my users the fault with the snake's movement that I found and partially corrected, and they have accepted that a complete fix would result in the development of a radically different program. With this in mind, they have each carried out testing of the program by playing the game as a normal player would, and filling in the following questionnaire.

Question number	Question	Answer (Y/N)	Comment
1	Do all the buttons and list boxes work?		
2	Are all the buttons/list boxes in appropriate places?		
3	Are the on screen prompts/statements useful, correct and sufficient?		
4	Does the system provide enough on screen instruction?		
5	Is the colour scheme appropriate?		
6	Does the system's look sufficiently match the original designs?		
7	Is it clear how you are to control the snake?		
8	Is it clear how you are to advance through the game?		
9	Are the error messages clear and useful?		
10	Is the game too easy?		
11	Is the game too difficult?		
12	Is the game screen size as desired?		
13	Does the application meet all the requirements of the initial specification?		
Please write here any comments that you have about the system that were not covered by the above questions			

I shall now list, next to each answer in parentheses, the frequency of each response made by my seven users. Any negative comments are noted, and will be discussed on the next page.

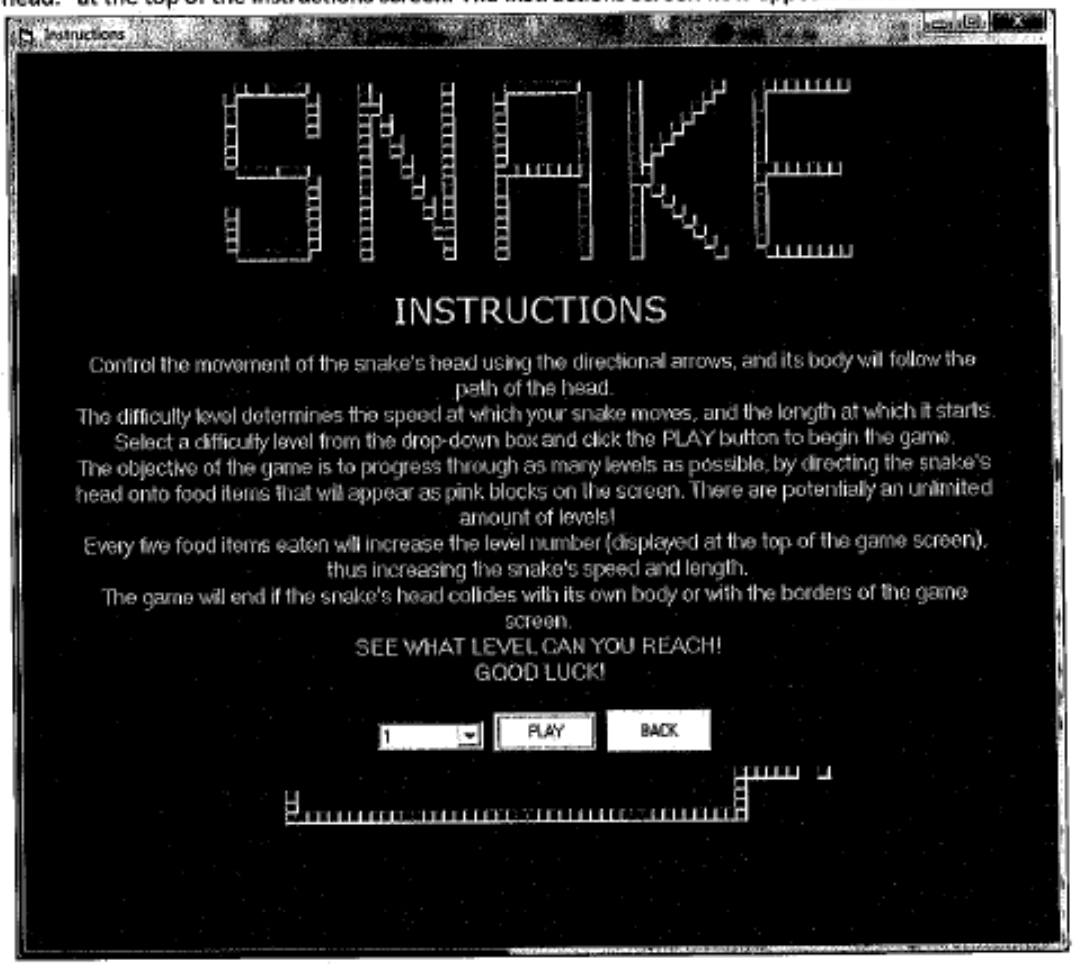
Question number	Question	Frequency of answer (Y/N)
1	Do all the buttons and list boxes work?	Yes (7) No (0)
2	Are all the buttons/list boxes in appropriate places?	Yes (3) No (4) – these four users noted that the instructions form appeared in a different place on the screen from the level selection form
3	Are the on screen prompts/statements useful, correct and sufficient?	Yes (7) No (0)
4	Does the system provide enough on screen instruction?	Yes (4) No (3) – these three users agreed that the player should be informed that there are more levels beyond level 10
5	Is the colour scheme appropriate?	Yes (7) – It looks 'retro' No (0)
6	Does the system's look sufficiently match the original designs?	Yes (7) No (0)
7	Is it clear how you are to control the snake?	Yes (4) No (3) – These three users agreed that it should be more clear, i.e. that it should say at the top of the instructions screen how you are to control the snake
8	Is it clear how you are to advance through the game?	Yes (7) No (0)
9	Are the error messages clear and useful?	Yes (7) No (0)
10	Is the game too easy?	Yes (0) No (7)
11	Is the game too difficult?	Yes (0) No (7)
12	Is the game screen size as desired?	Yes (7) No (0)
13	Does the application meet all the requirements of the initial specification?	Yes (7) No (0)

There were few negative comments from my users, but the following were made:

1) The player should be informed that there are more levels beyond level 10
2) It should say at the beginning of the instructions screen how you are to control the snake
3) The instructions form appears at a different place on the screen from the initial menu screen – it should instead appear in the same place

I have dealt with each comment individually, as follows:

1 and 2: I fixed both of these issues by saying at the end of the third line on the instructions screen "There are potentially an unlimited amount of levels!", and by inserting the line "Control the movement of the snake's head using the directional arrows, and its body will follow the path of the head." at the top of the instructions screen. The instructions screen now appears as below.



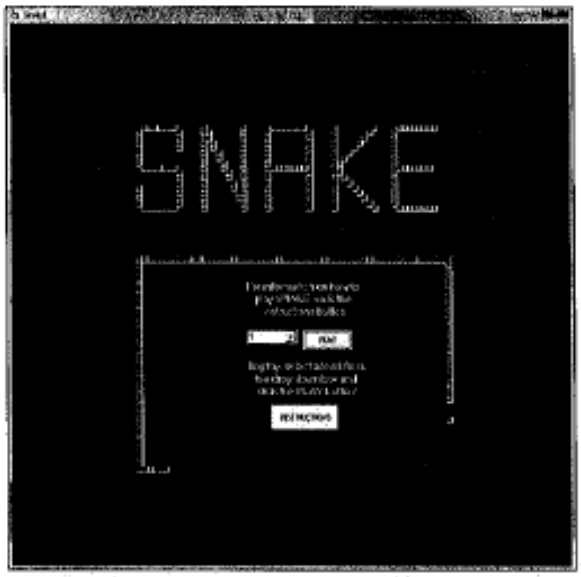
3: This problem was simply resolved by changing the StartUpPosition property of frmInstructions from 3 (windows default) to 2 (centre screen).

The changes I have made resulting from user testing have been purely issues of aesthetics, and as such no further testing needs to be carried out on the program to ascertain functionality.

Annotated code – form named frmSnake

Below is the final program code for Snake, including the changes that resulted from testing.

This is the form named frmSnake in its two states:



These are the objects that it initially contains: imgSnake(0 to 4), cmdPlay, cmbLevel, lblInstructions1, lblInstructions2, cmdInstructions, imgLogo1, imgLogo2 and lblLevel.

Declarations for frmSnake variables – these are written at the top of the code, descriptions of each variable are written after the apostrophe

```
Option Explicit 'forces me to declare all variables
Dim snake(9999) As SnakeBit   'stores x and y values, and direction facing for each snake bit
Dim valid As Boolean    'used to flag up if a food item is to be placed on top of a snake bit so that this can be avoided
Dim i As Integer            'constants
Dim k As Integer
Dim length As Integer  'stores length of snake and is equal to number of snake bits - 1
Dim foodTop As Single     'stores top value of food item
Dim foodLeft As Single    'stores left value of food item
Dim facing(-84 To 999999) As Byte      'stores the direction the head was facing (the value stored) on a particular move (the index)
Dim moves As Long             'stores amount of times the timer is called
Dim keyPressed As Single  'stores keycode value of directional button pressed
Dim foodCount As Byte       'stores how many food items are eaten, up to every 5th item eaten
Dim newBits As String       'stores the index of each new bit that arises after a level up
```

Form load and button click subroutines

```
Private Sub cmdInstructions_Click()
   frmSnake.Visible = False
   frmInstructions.Visible = True
End Sub
```

This hides the level selection screen and displays the instructions screen when cmdInstructions is clicked

```
Private Sub Form_Load()
   cmbLevel.Text = 1
End Sub
```

This ensures that the starting value of cmbLevel is 1

```
Private Sub cmdLevel_Click()
   level = cmbLevel.Text
   Call StartGame
End Sub
```

When cmdLevel is clicked, this sets 'level' to the value selected by the player and then calls the subroutine named StartGame

StartGame subroutine – (this is a public subroutine, so that it can be called from frmInstructions)

```
Public Sub StartGame()
   lblInstructions.Visible = False
   lblInstructions2.Visible = False
   lblLevel.Visible = True
   cmdLevel.Visible = False
   cmbLevel.Visible = False
   cmdInstructions.Visible = False
   cmdLevel.Enabled = False
   cmbLevel.Enabled = False
   cmdInstructions.Enabled = False
   frmSnake.BackColor = &H0&
   imgLogo1.Visible = False
   imgLogo2.Visible = False
```

This hides and disables the buttons, combo box, labels and images particular to the level selection screen and displays the level display and makes the background of the form black

```
   length = 30 + ((level - 1) * 6)
   For k = 0 To (length - 1)
      If k > 4 Then Load imgSnake(k)
      imgSnake(k).Visible = True
      imgSnake(k).Top = 300
      snake(k).Y = 300
      imgSnake(k).Left = 300 - (10 * k)
      snake(k).X = 300 - (10 * k)
      snake(k).Direction = 2
   Next k
```

This sets the length of the snake according to the level chosen, storing this in 'length', loads images of snake bits if they aren't already loaded, makes them visible, positions them, sets x and y co-ordinates of 'snake' and sets directional values of 'snake' to 2 (right)

```
For i = -84 To 0
      facing(i) = 2
   Next i

   moves = 0
```

This initialises 'facing' so that the default value for 'facing' for all starting snake bits is 2

```
    foodCount = 0
    lblLevel.Caption = "LEVEL: " & level
    tmrMoveSnake.Enabled = True

    Call PlaceFood

    tmrMoveSnake.Interval = 110 - (level * 10)

    Call tmrMoveSnake_Timer
End Sub
```

This initialises the variables 'moves' and 'foodCount', sets the level display end enables the timer that controls snake movement

This adjusts the interval of the timer depending on the level chosen

PlaceFood subroutine

```
Private Sub PlaceFood()
    Do
        Randomize
        valid = True
        foodTop = Int(Rnd * 89) * 10
        foodLeft = Int(Rnd * 89) * 10
        For k = 0 To (length - 1)
            If foodTop = snake(k).Y And foodLeft = snake(k).X Then
                valid = False
                Exit For
            End If
        Next k
    Loop Until valid = True
    imgFood.Visible = True
    imgFood.Top = foodTop
    imgFood.Left = foodLeft
End Sub
```

This randomises the Random function in visual basic 6, initialises 'valid' with a value of true and obtains random x and y values for the 'foodLeft' and 'foodTop' that are multiples of 10 and between 0 and 890

This ensures that food isn't placed on top of a snake bit, by looping through all snake bits and determining whether the foodTop/foodLeft co-ordinates are the same as the co-ordinates of any one snake bit. If they are, then 'valid' is false and new co-ordinates are randomly chosen and the process continues until 'valid' remains true

This positions the food image with the co-ordinates now stored in 'foodTop' and 'foodLeft' and ensures that it is visible

Timer event subroutine

```
Private Sub tmrMoveSnake_Timer()
    moves = moves + 1
    Call CheckSnakeCollisionAndMoveSnakeBody
    Call MoveSnakeHead
    Call CheckWallCollision
End Sub
```

This increments the amount in 'moves' to store the amount of moves made, and calls the subroutine to move the snake's body and check for a snake collision, then calls the subroutine to move the snake's head, then calls the subroutine to check if there is a wall collision

CheckSnakeCollisionAndMoveSnakeBody subroutine

```
Private Sub CheckSnakeCollisionAndMoveSnakeBody()
    For i = 1 To (length - 1)
        If snake(0).Y = snake(i).Y And snake(0).X = snake(i).X Then
            tmrMoveSnake.Enabled = False
            MsgBox ("Snake Dead!")
```

This checks for a snake collision by looping through the snake bits that follow the head and determining whether the snake's head has matching co-ordinates. If a collision is detected, the timer is disabled, the snake is made invisible and all but 5 snake bits are unloaded, an error message is produced and EndGame is called

```
        For k = 0 To (length - 1)
            imgSnake(k).Visible = False
            If k > 4 Then Unload imgSnake(k)
        Next k
        Call EndGame
        Exit Sub
    End If
    If InStr(newBits, " " & i & " ") = 0 Then
        snake(i).Direction = facing(moves - i)
    End If
    If i = length - 1 Then newBits = ""
    Select Case snake(i).Direction
        Case 1
        snake(i).Y = snake(i).Y + 10
        imgSnake(i).Top = snake(i).Y
        Case 2
        snake(i).X = snake(i).X + 10
        imgSnake(i).Left = snake(i).X
        Case 3
        snake(i).Y = snake(i).Y - 10
        imgSnake(i).Top = snake(i).Y
        Case 4
        snake(i).X = snake(i).X - 10
        imgSnake(i).Left = snake(i).X
    End Select
  Next i
End Sub
```

This alters the snake bits direction stored in snake().Direction so that they face the direction that the head was facing on a move numbered (moves- number of snake bit), provided the snake bits have not just been added to the snake as a result of a level increase, in which case the index of the snake bit will be stored in 'newBits'. It then ensures that 'newBits' is emptied when all new bits have been moved once, so that they act like normal bits

This moves each bit of the snake's body by 10 pixels according to the direction it is facing, by increasing/decreasing the values stored in 'snake(i).X' and setting this value to be imgSnake(i).Left if the snake bit is facing right or left (ie when 'snake(i).Direction' = 2 or 4) or increasing/decreasing the values stored in 'snake(i).Y' and setting this value to be imgSnake(i).Top if the snake bit is facing up or down (ie when 'snake(i).Direction' = 1 or 3)

)

MoveSnakeHead subroutine

```
Private Sub MoveSnakeHead()
  Select Case snake(0).Direction
    Case 1
      snake(0).Y = snake(0).Y + 10
      imgSnake(0).Top = snake(0).Y
      facing(moves) = 1
    Case 2
      snake(0).X = snake(0).X + 10
      imgSnake(0).Left = snake(0).X
      facing(moves) = 2
    Case 3
      snake(0).Y = snake(0).Y - 10
      imgSnake(0).Top = snake(0).Y
      facing(moves) = 3
    Case 4
      snake(0).X = snake(0).X - 10
      imgSnake(0).Left = snake(0).X
      facing(moves) = 4
  End Select

End Sub
```

This moves the snake's head by 10 pixels according to the direction it is facing, by increasing/decreasing the values stored in 'snake(0).X' and setting this value to be imgSnake(0).Left if the head is facing right or left (ie when 'snake(0).Direction' = 2 or 4) or increasing/decreasing the values stored in 'snake(0).Y' and setting this value to be imgSnake(0).Top if the head is facing up or down (ie when 'snake(0).Direction' = 1 or 3). It then stores in 'facing' the direction (the value stored) on this particular move (the index)

CheckWallCollision subroutine

```
Private Sub CheckWallCollision()
  If snake(0).X >= 900 Or snake(0).X < 0 Or snake(0).Y >= 900 Or snake(0).Y < 0 Then
    tmrMoveSnake.Enabled = False
    MsgBox ("Snake Dead!")
    For i = 0 To (length - 1)
      imgSnake(i).Visible = False
      If i > 4 Then Unload imgSnake(i)
    Next i
    Call EndGame
  End If
Call CheckFoodCollision
End Sub
```

This detects when the snake's head collides with the borders of the game screen, by determining whether X the co-ordinates of imgSnake(0) that are stored in snake(0).X are >= 900 or <0, or whether the Y co-ordinates stored in snake(0).Y are >=900 or <0.
If this condition is true, then the timer is disabled, an error message is displayed, all the snake images are made invisible and all apart from 4 are unloaded, then EndGame is called.

EndGame subroutine

```
Private Sub EndGame()
    imgLogo1.Visible = True
    imgLogo2.Visible = True
    frmSnake.BackColor = &H808080
    imgFood.Visible = False
    lblInstructions.Visible = True
    lblInstructions2.Visible = True
    lblLevel.Visible = False
    cmdLevel.Visible = True
    cmbLevel.Visible = True
    cmdLevel.Enabled = True
    cmbLevel.Enabled = True
    cmdInstructions.Visible = True
    cmdInstructions.Enabled = True
End Sub
```

This is the subroutine that is called when a game of snake ends. It hides the game screen and displays the level selection screen by making the logo images visible, setting the background colour of frmSnake to grey, making the food image invisible, making the instructions labels visible, making the level display invisible and making the combo box and buttons of the level selection screen visible and enabled.

CheckFoodCollision subroutine

```
Private Sub CheckFoodCollision()
  If snake(0).Y = foodTop And snake(0).X = foodLeft Then
    If foodCount <> 4 Then
      foodCount = foodCount + 1
    Else
      foodCount = 0
      level = level + 1
      lblLevel.Caption = "LEVEL: " & level
      If level <= 10 Then tmrMoveSnake.Interval = 110 - (level * 10)
      For k = length To (length + 5)
        Load imgSnake(k)
        imgSnake(k).Visible = True
        newBits = newBits & " " & k
        Select Case snake(length - 1).Direction
          Case 1
            imgSnake(k).Top = snake(k - 1).Y - 10
            snake(k).Y = imgSnake(k).Top
            imgSnake(k).Left = (snake(k - 1).X)
            snake(k).X = imgSnake(k).Left
            snake(k).Direction = 1
            facing(moves - k) = snake(k).Direction
          Case 2
            imgSnake(k).Top = (snake(k - 1).Y)
            snake(k).Y = imgSnake(k).Top
            imgSnake(k).Left = snake(k - 1).X - 10
            snake(k).X = imgSnake(k).Left
            snake(k).Direction = 2
            facing(moves - k) = snake(k).Direction
          Case 3
            imgSnake(k).Top = (snake(k - 1).Y) + 10
            snake(k).Y = imgSnake(k).Top
            imgSnake(k).Left = snake(k - 1).X
            snake(k).X = imgSnake(k).Left
            snake(k).Direction = 3
            facing(moves - k) = snake(k).Direction
          Case 4
            imgSnake(k).Top = snake(k - 1).Y
            snake(k).Y = imgSnake(k).Top
            imgSnake(k).Left = (snake(k - 1).X + 10)
            snake(k).X = imgSnake(k).Left
            snake(k).Direction = 4
            facing(moves - k) = snake(k).Direction
        End Select
      Next k
```

This checks for a collision with food by determining whether the X co-ordinates of imgSnake(0) that are stored in snake(0).X are the same as the X co-ordinates of imgFood stored in 'foodLeft' and whether the Y co-ordinates stored in snake(0).Y are the same as the Y co-ordinates of imgFood stored in 'foodTop'.

If this is true, and if 'foodCount' is <> 4, then there is no level up and 'foodCount' is simply incremented.

If 'foodCount' = 4 then it gets reset to 0, 'level' gets incremented, the level display in lblLevel is refreshed and the timer interval is changed if necessary.

Then this loads images of the six new snake bits , makes them visible ar stores their indexes in 'newBits'.

This positions each new snake image behind the tail (the end snake bit) according to the direction the end is facing . It does this by giving each new snake image the top values of the snake bit before it and the left values of the snake bit before it minus 10 if the tail is facing right, or plus 10 if the tail is facing left; or by giving each new snake bit the left values of the snake bit before it and the top values of the snake bit before it minus 10 if the tail is facing up, or plus 10 if the tail is facing down.
It then updates the X and Y co-ordinates stored in snake(k).X and snake(k).Y to the left and top values of imgSnake(k), assigns the directional value of the end snake bit to the new snake bit, and updates facing(moves-k) to store snake(k).Direction so that the new snake bit follows the tail

```
      length = length + 6
    End If
    Call PlaceFood
  End If
End Sub
```

This updates the value stored in length (if the level has been incremented) and then calls Placefood.

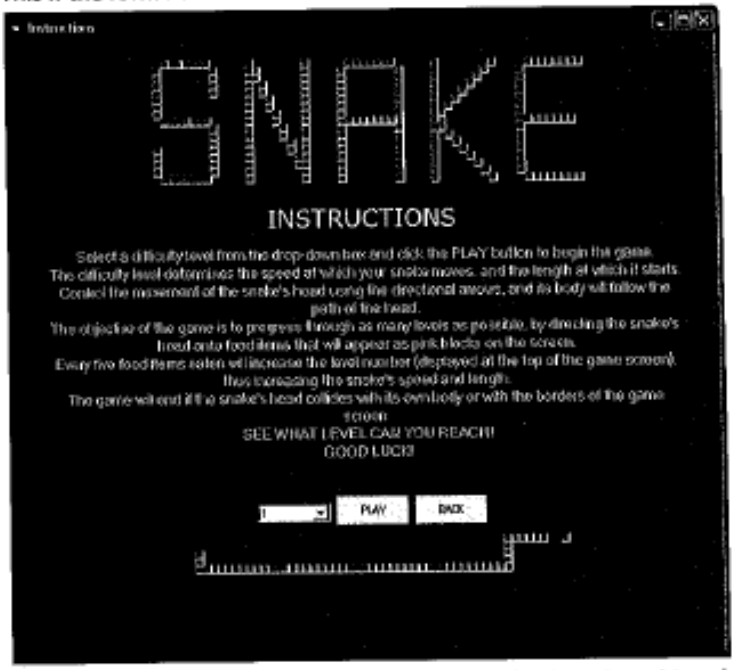
Key press event subroutine

```
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
  keyPressed = KeyCode
  Select Case keyPressed
    Case 37
    If snake(1).Y <> snake(0).Y Or snake(1).Y <> snake(0).Y Then snake(0).Direction = 4
    Case 38
    If snake(1).X <> snake(0).X Or snake(1).X <> snake(0).X Then snake(0).Direction = 3
    Case 39
    If snake(1).Y <> snake(0).Y Or snake(1).Y <> snake(0).Y Then snake(0).Direction = 2
    Case 40
    If snake(1).X <> snake(0).X Or snake(1).X <> snake(0).X Then snake(0).Direction = 1
  End Select
End Sub
```

This is the subroutine called when a key on the keyboard is pressed. It sets 'keyPressed' to the key code value of the key that was pressed, and then adjusts the direction that the snake's head is facing according to which directional button was pressed, and ensures that the snake can only change direction to down or up if 'snake(1).Y' is not equal to 'snake(0).Y' , and the snake can only change direction to left or right if 'snake(1).X' is not equal to 'snake(0).X', so that the snake cannot go back inside itself.

Annotated code – form named frmInstructions

This if the form name frmInstructions:



These are the objects that it contains: imgLogo, imgLogo3, cmbLevel, cmdLevel, cmdBack, lblInstructions3 and lblInstructions4

There are no declarations in the code for frmSnake. The code consists only of form load and button click subroutines.

Form load and button click subroutines

```
Private Sub Form_Load()
  cmbLevel.Text = 1
  lblInstructions4.Caption = "Select a difficulty level from the drop-down box and click the PLAY
  button to begin the game." & vbNewLine & "The difficulty level determines the speed at which
  your snake moves, and the length at which it starts." & vbNewLine & "Control the movement of
  the snake's head using the directional arrows, and its body will follow the path of the head." &
  vbNewLine & "The objective of the game is to progress through as many levels as possible, by
  directing the snake's head onto food items that will appear as pink blocks on the screen." &
  vbNewLine & "Every five food items eaten will increase the level number (displayed at the top of
  the game screen), thus increasing the snake's speed and length." & vbNewLine & "The game will
  end if the snake's head collides with its own body or with the borders of the game screen." &
  vbNewLine & "SEE WHAT LEVEL CAN YOU REACH!" & vbNewLine & "GOOD LUCK!"
End Sub
```

This ensures that the default value of cmbLevel.Text is 1, and fills the caption of lblInstructions4 with the instructions for playing the game when the form loads

```
Private Sub cmdLevel_Click()
   level = cmbLevel.Text
   frmSnake.Visible = True
   frmInstructions.Visible = False
   Call frmSnake.StartGame
End Sub
```

When cmdLevel is clicked, this sets the level value to the value selected by the player in cmbLevel, and then makes frmSnake visible and makes frmInstructions invisible, and then calls the public subroutine called StartGame

```
Private Sub cmdBack_Click()
   frmSnake.Visible = True
   frmInstructions.Visible = False
End Sub
```

When cmdBack is clicked, this makes frmSnake visible and makes frmInstructions invisible

Annotated code – module named mdlSnake

This module exists only to make 'level' a public variable and to define the UDT SnakeBit

Declarations of module named mdlSnake

```
Public level As Byte
```

Level is used to store the curent level

```
Type SnakeBit   ' establishing data type for each snake 'bit'
   Direction As Byte   'direction the snake bit is facing (1 = down, 2 = right, 3 = up, 4 = left)
   X As Long   ' X coordinate of snake bit
   Y As Long   ' Y coordinate of snake bit
End Type
```

This creates the data type Snake bit, that consists of the variables direction (that stores the direction a snake bit is facing where 1 = down, 2 = right, 3 = up, and 4 = left), X (that stores the X xo-ordinate of a snake bit) and Y (that stores the Y co-ordinate of a snake bit)

Section 4 - Additional documentation

The finished snake program will be distributed online. It will be uploaded to the website vbgamer.com, so the user will be required to download the program in order to play it. The following documentation completes the guide to the program.

1

Contents

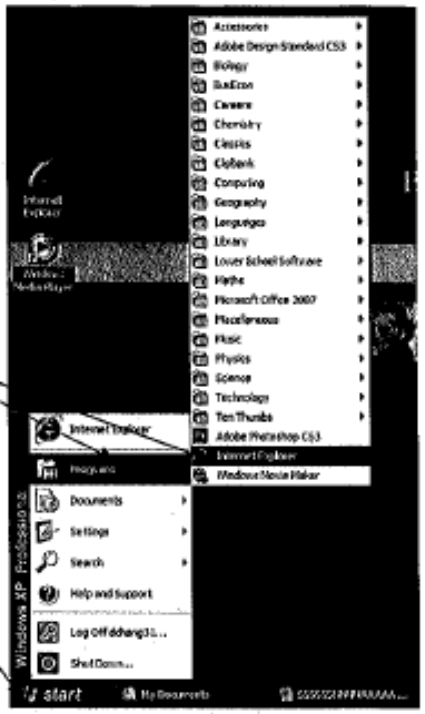
1 Download and installation....2
2 Instructions....5
2.1 Starting the application....5
2.2 Navigation....6
2.2.1 Initial level selection screen....6
2.2.2 Instructions screen....7
3 How to play the game....8
3.1 Level selection....8
3.2 Controlling the snake....9
3.3 Level progression....10
4 Troubleshooting....14
4.1 Download and installation issues....14
4.2 General issues....14
5 Glossary....15

1 – Download and installation

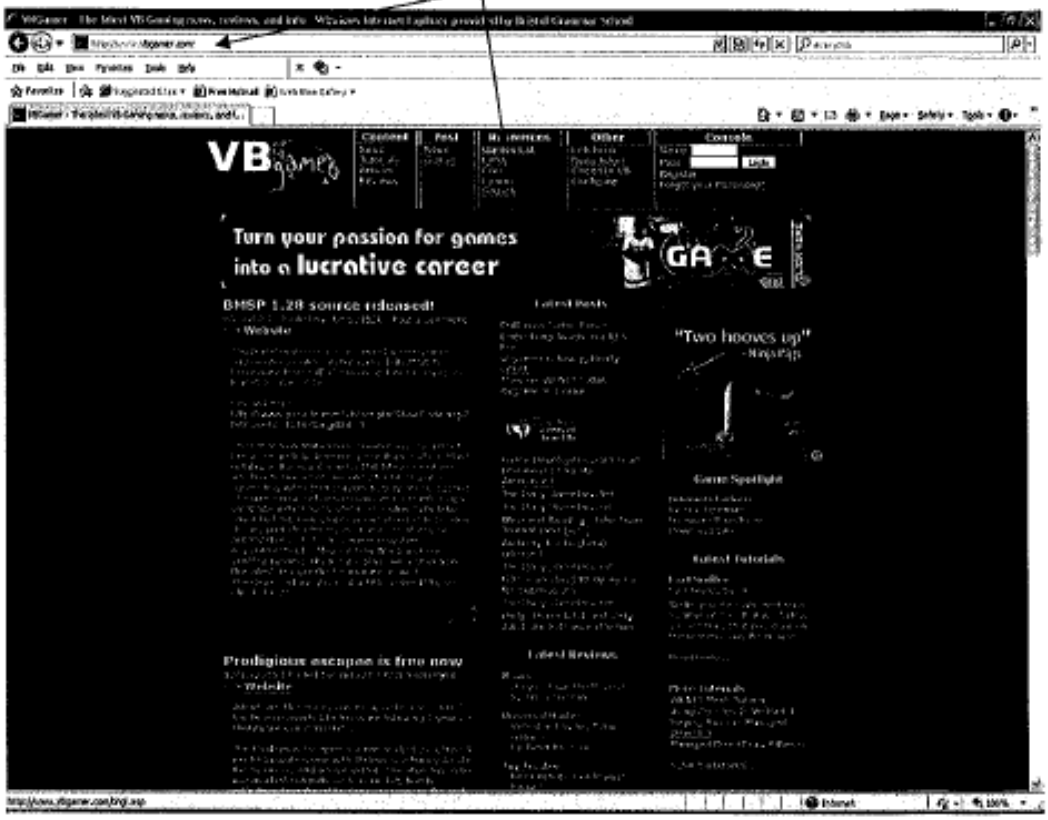
2

To download Snake:

- Click on Start
- Click on All Programs
- Select your web browser

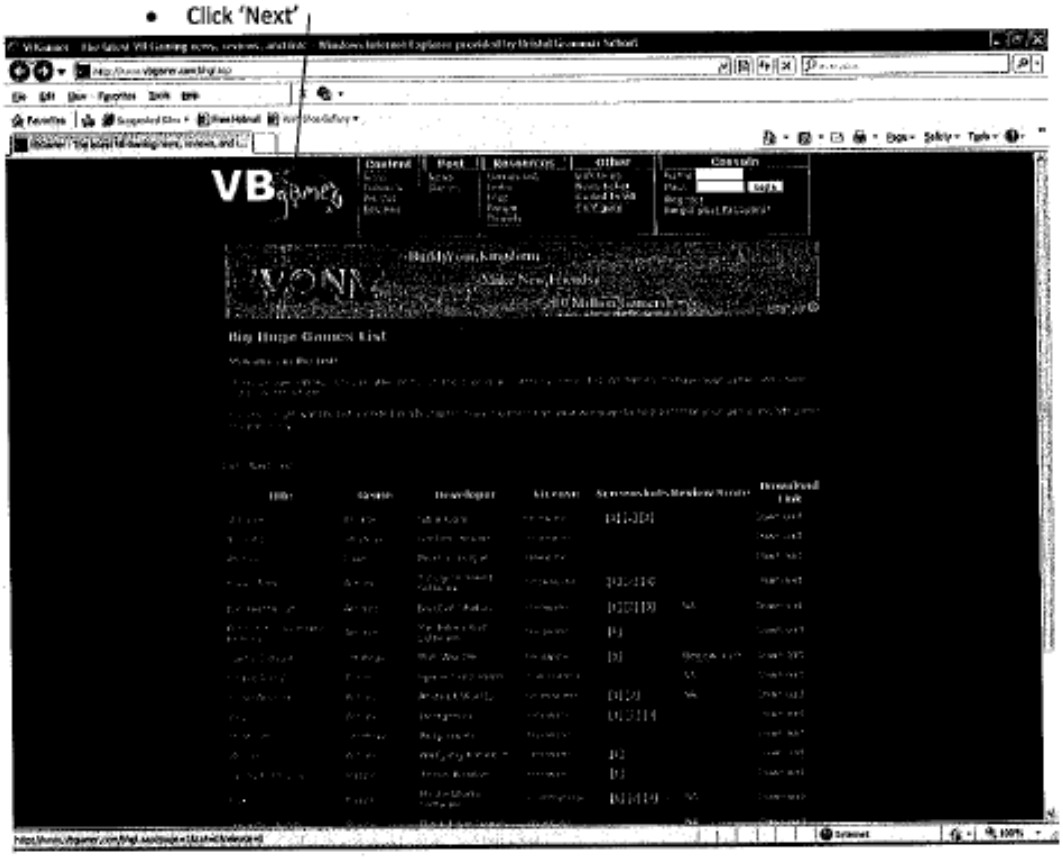


- Type www.vbgamer.com into the address bar and press enter
- Click 'Games List' under the Resources tab

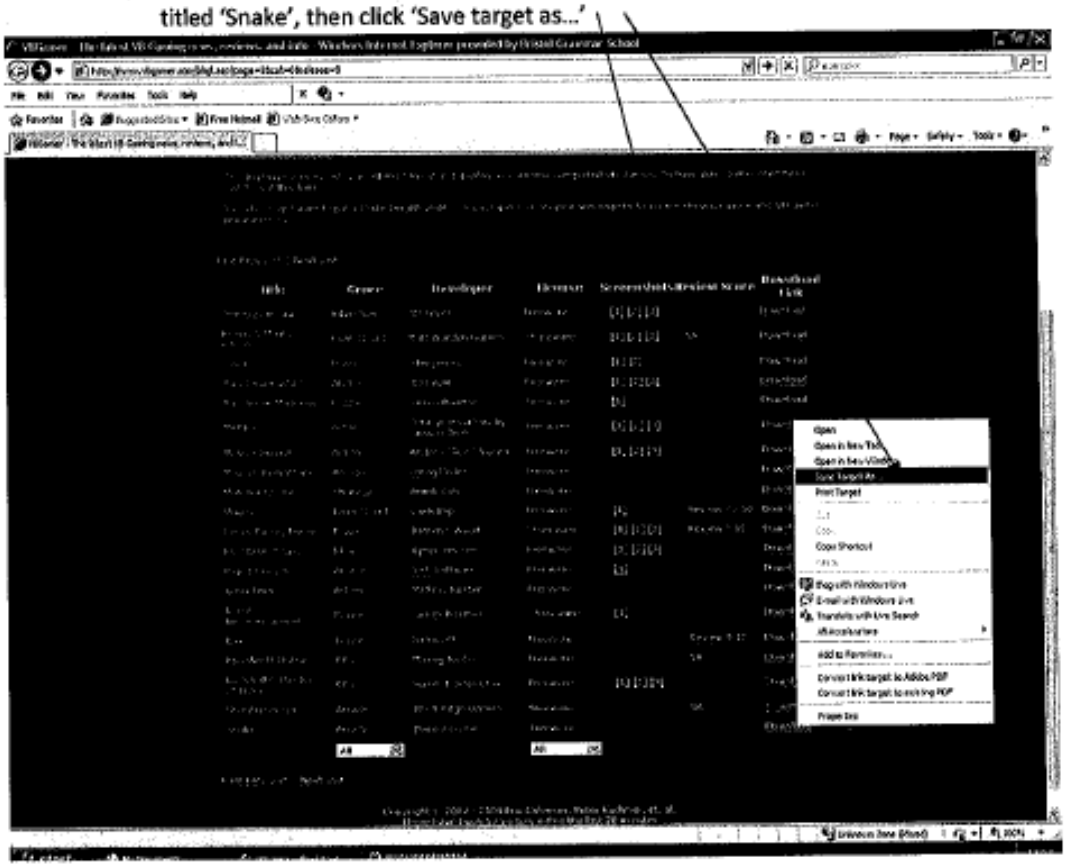


3

- Click 'Next'

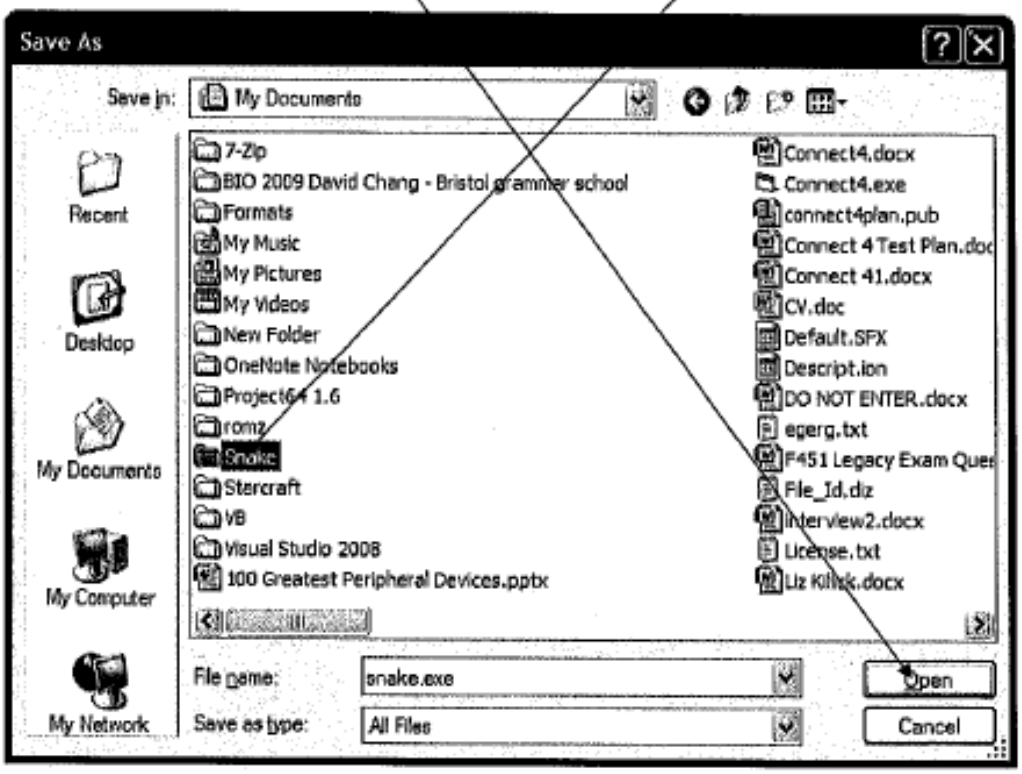


- Scroll to the bottom of the page and right click 'Download' on the row of the game titled 'Snake', then click 'Save target as...'

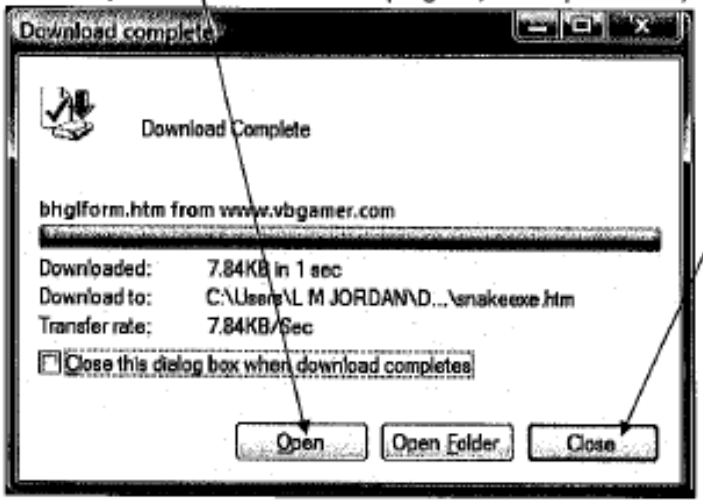


4

- Select the folder in which you wish the game to be saved
- Click 'Open' or press enter



- If you wish to run the snake program, click 'open'. If not, click 'close'.



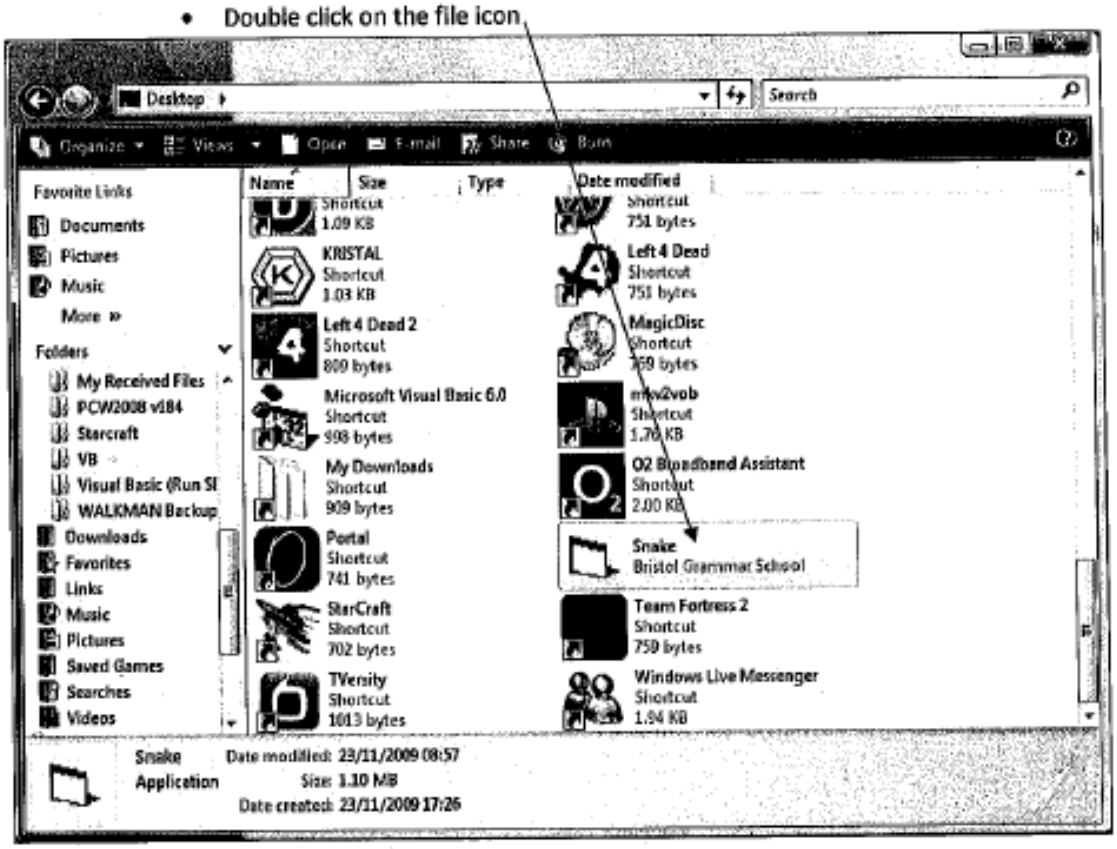
5

2 – Instructions

2.1 – Starting the application

To start the snake program:

- Navigate to where you saved the program
- Double click on the file icon

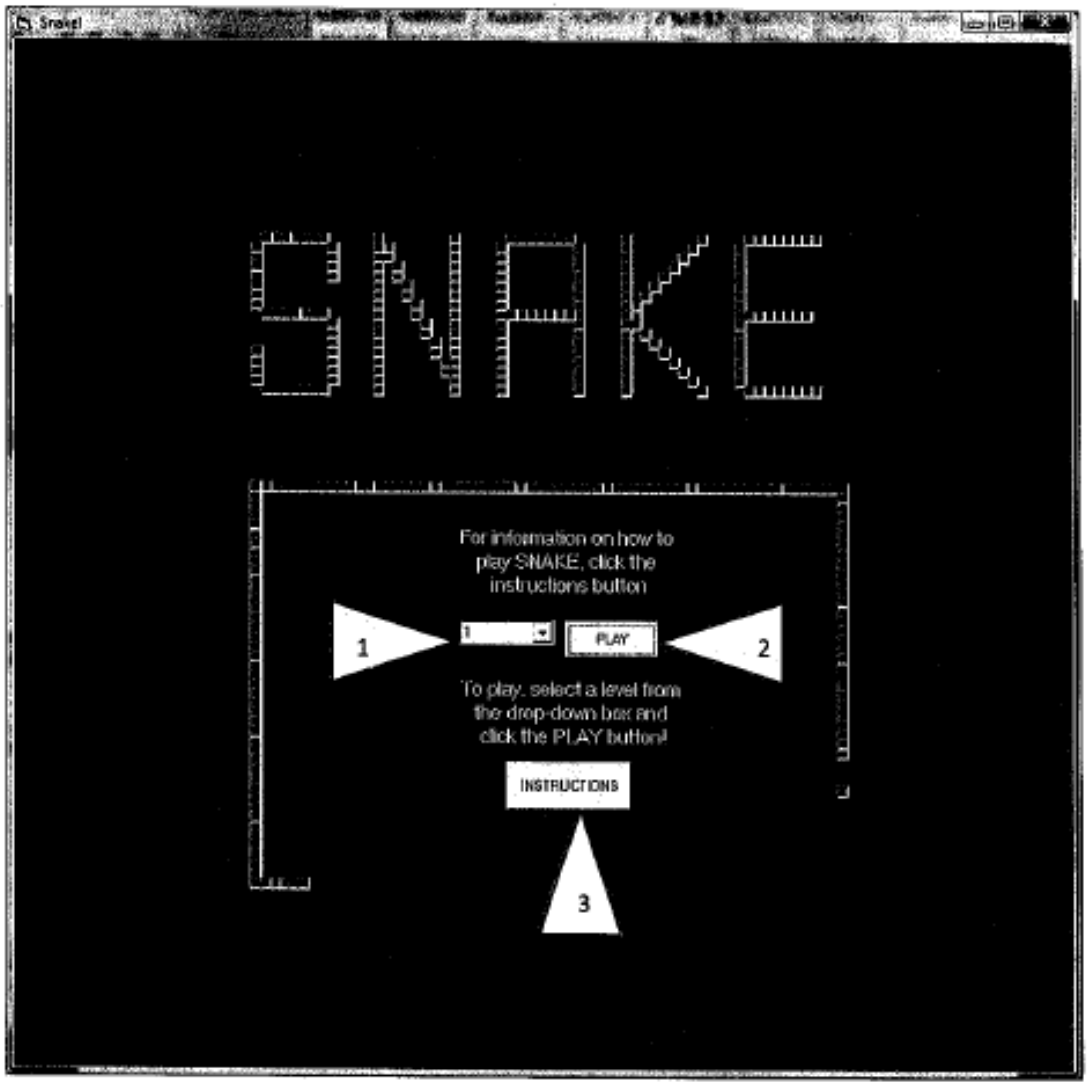


2.2 – Navigation

6

2.2.1 – Initial level selection screen

This is the initial level selection screen of the Snake program. Each object that can be interacted with has a corresponding adjacent numbered triangle, with its function explained below.

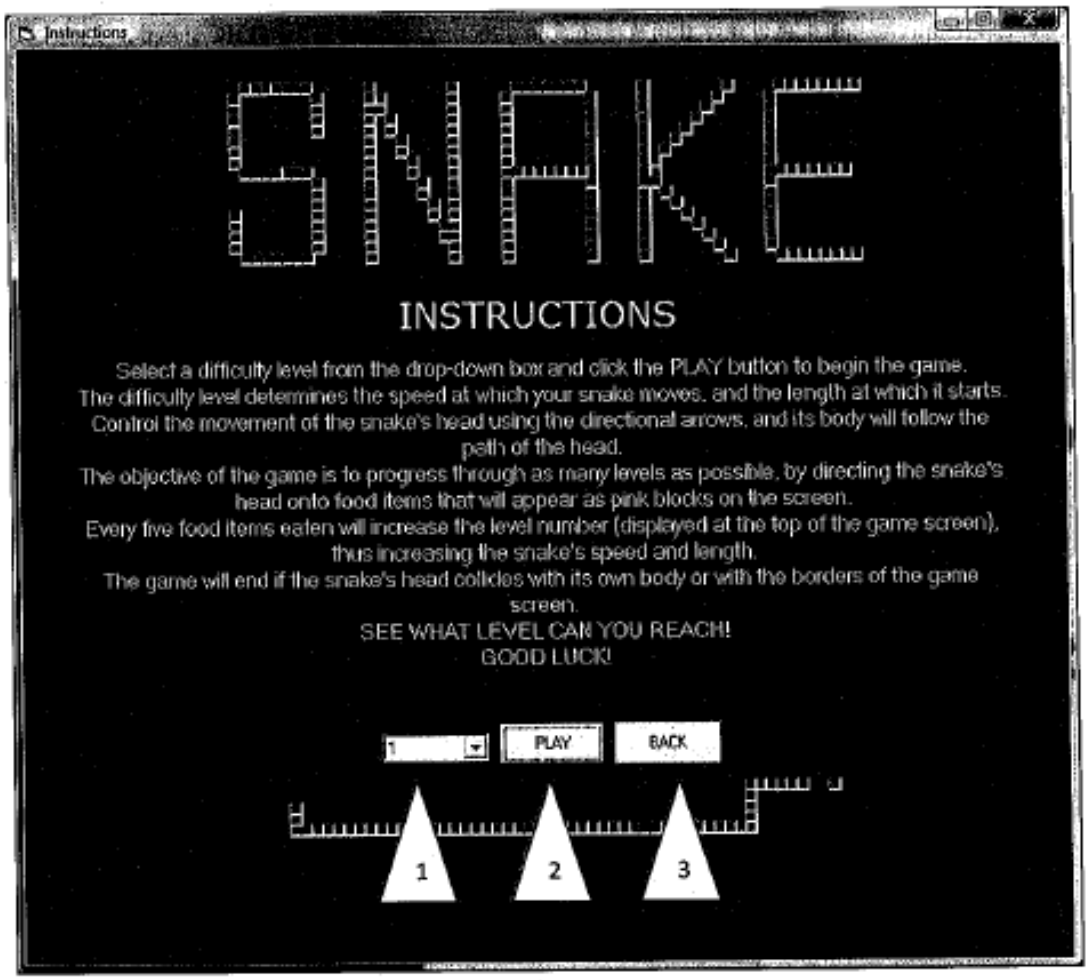


1) This is the level selection box – click on the arrow at the end of the box to select from a drop-down list the level number at which you wish to begin playing the game.
2) This is the 'Play' button – after selecting a level number from the drop down list contained in the level selection box, clicking this button will begin a new game
3) This is the 'Instructions' button – clicking this button will open a new form that displays instructions on how to play the game.

2.2.2 – Instructions screen

7

This is the instructions screen of the Snake program. Each object that can be interacted with has a corresponding adjacent numbered triangle, with its function explained below.

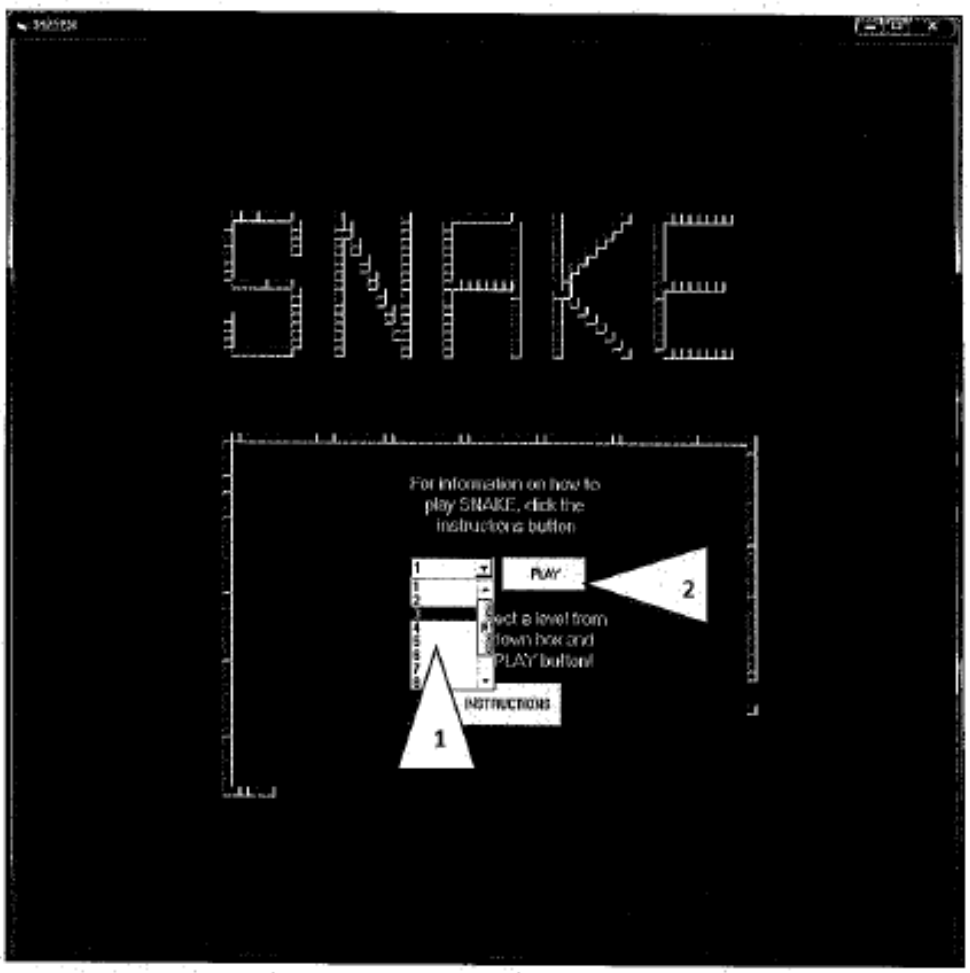


1) This is the level selection box – click on the arrow at the end of the box to select from a drop-down list the level number at which you wish to begin playing the game.
2) This is the 'Play' button – after selecting a level number from the drop down list contained in the level selection box, clicking this button will begin a new game
3) This is the 'Back' button – clicking this button will return you to the initial level selection screen

3 – How to play the game

3.1 - Level selection

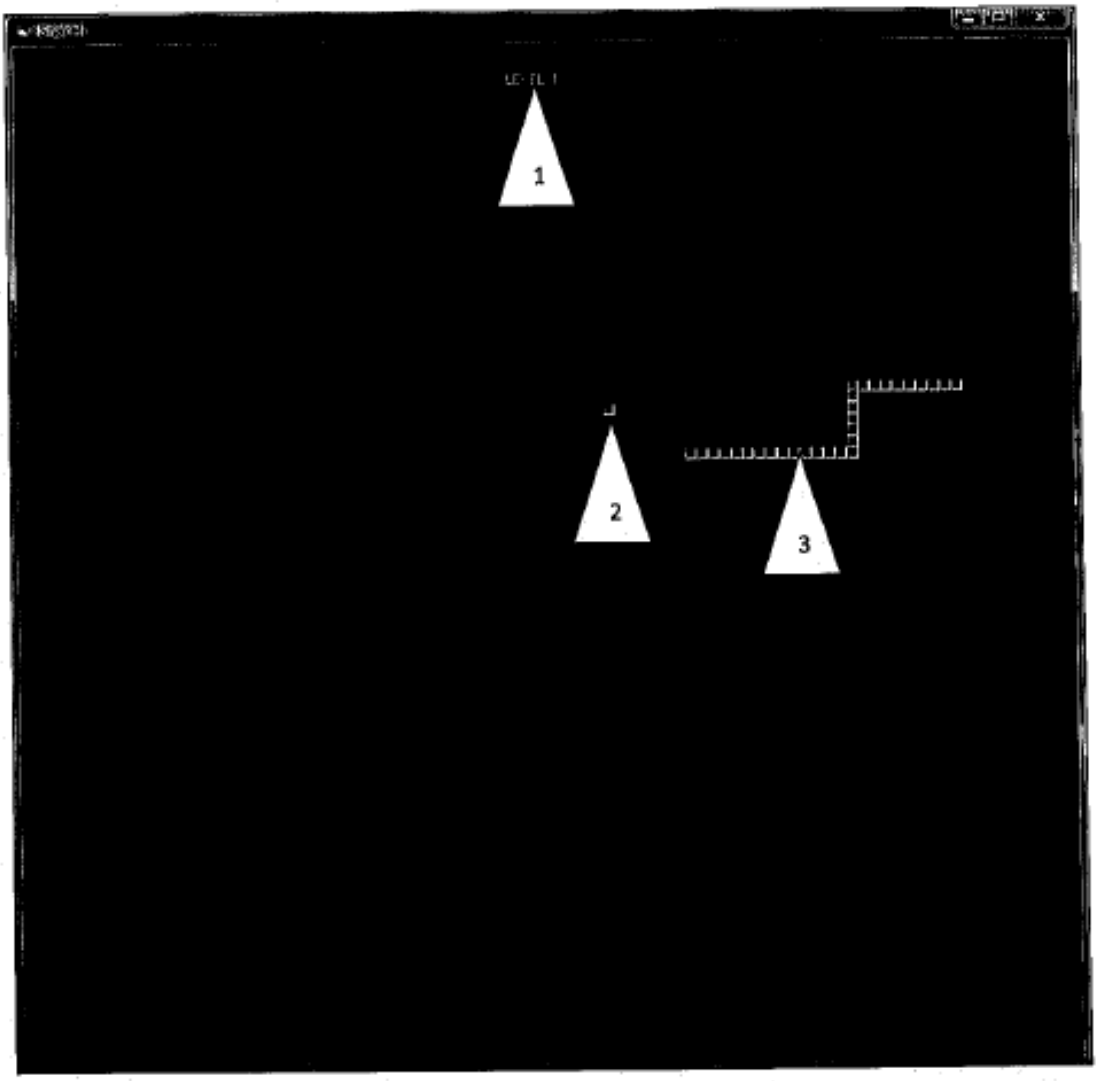
To play snake, first you must select a difficulty level from the drop-down box (1) and click the PLAY button (2) on the initial level selection screen. The difficulty level determines the speed at which your snake moves, and the length at which it starts. The snake starts at a length of 30 'bits', and increases in length by 6 'bits' every time you reach a new level.



3.2 - Controlling the snake

Control the movement of the snake's head using the directional arrows on the keyboard – if you press the up directional arrow, the snake's head will move upwards, if you press the right directional arrow, it will move right etc. However, you can only move right or left when the snake is moving up or down, and you can only move up or down when the snake is moving right or left. The snake's body will follow the path of the head. The game screen is displayed on the next page.

Each object on the game screen is here identified with a numbered triangle.



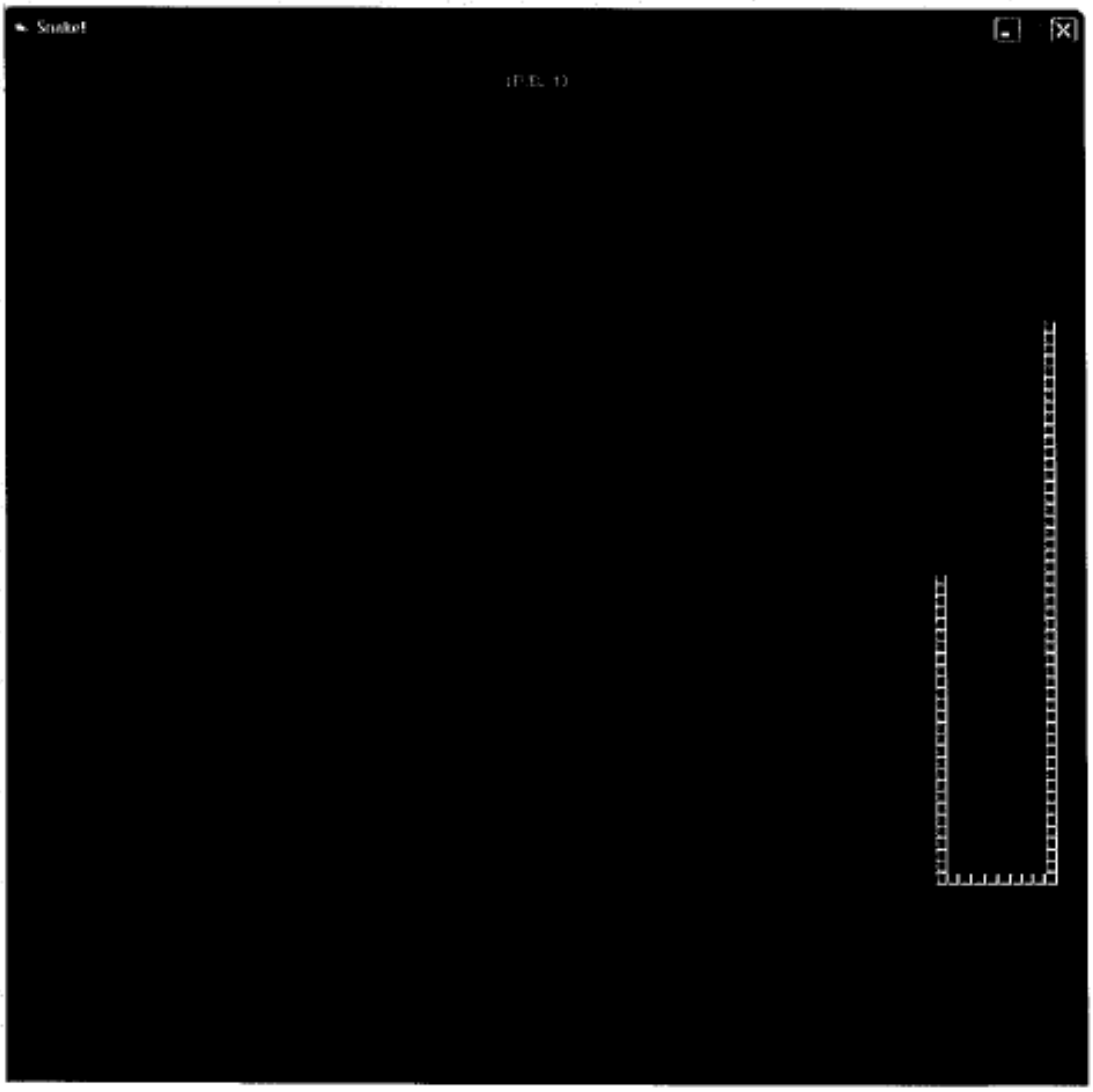
1) This is the level display. It displays the number of the level that you are currently playing
2) This is a food item
3) This is the snake

3.3 - Level progression

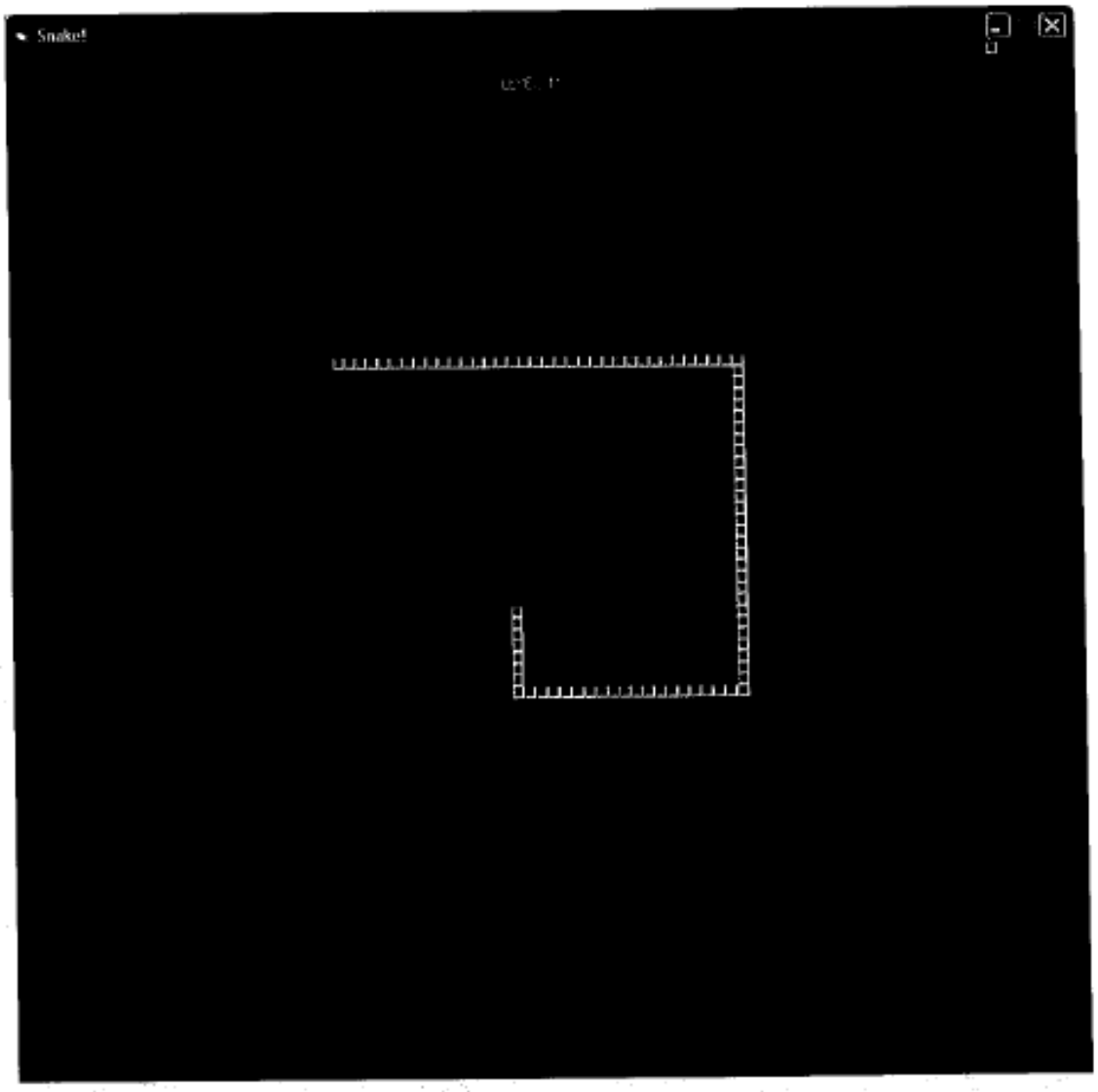
10

The objective of the game is to progress through as many levels as possible, by directing the snake's head onto food items that will appear as pink blocks on the screen. Every five food items eaten will increase the level number (displayed at the top of the game screen), thus increasing the snake's speed and length. However, at levels above level 10 the speed remains constant, but the snake will continue to increase in length.

This shows, on level 10, the snake's head coming into contact with a food item. This causes the snake to 'eat' the food, and another food item will then be placed on the screen.



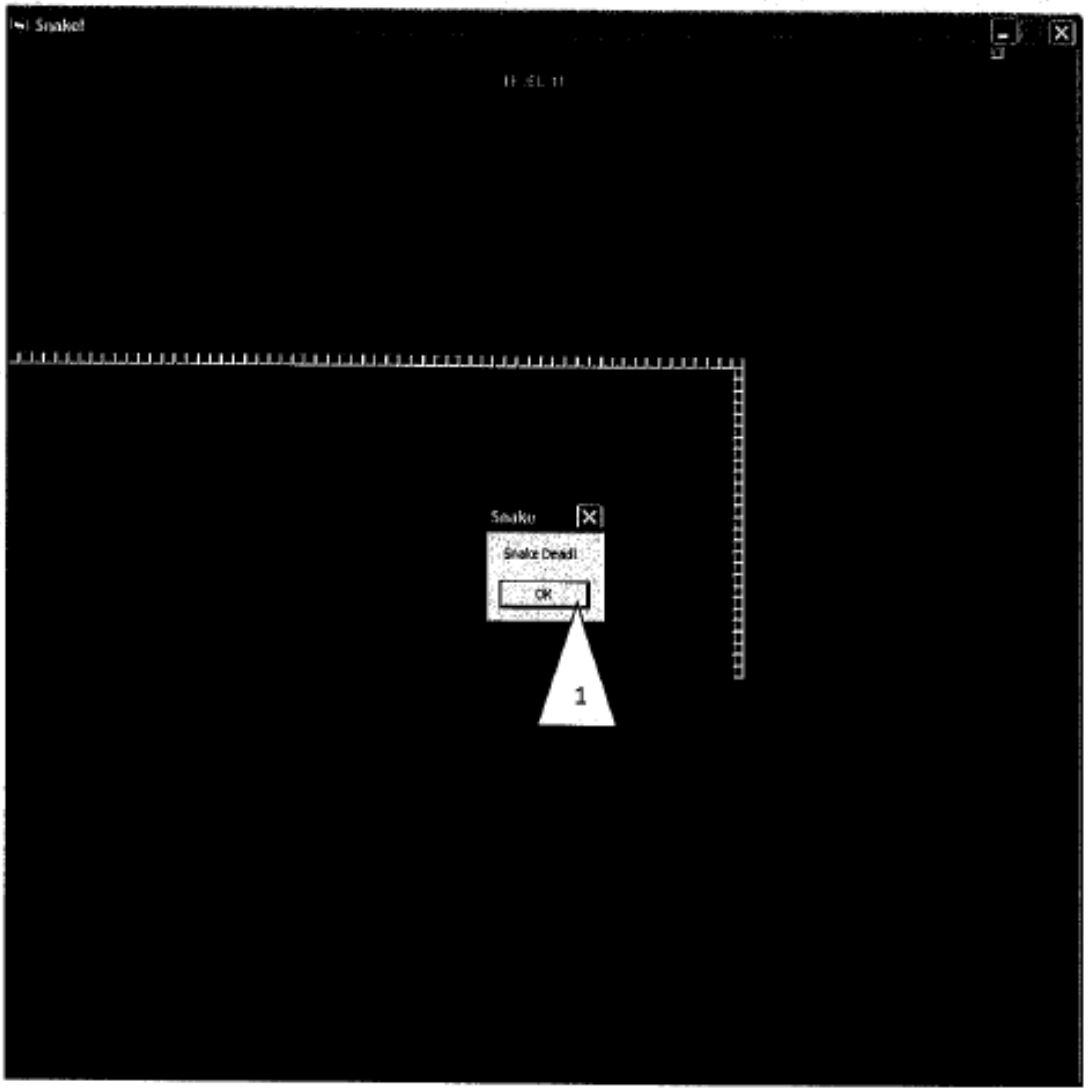
This shows the snake after it has progressed to the next level, having eating five food items. Note its increased length.



12

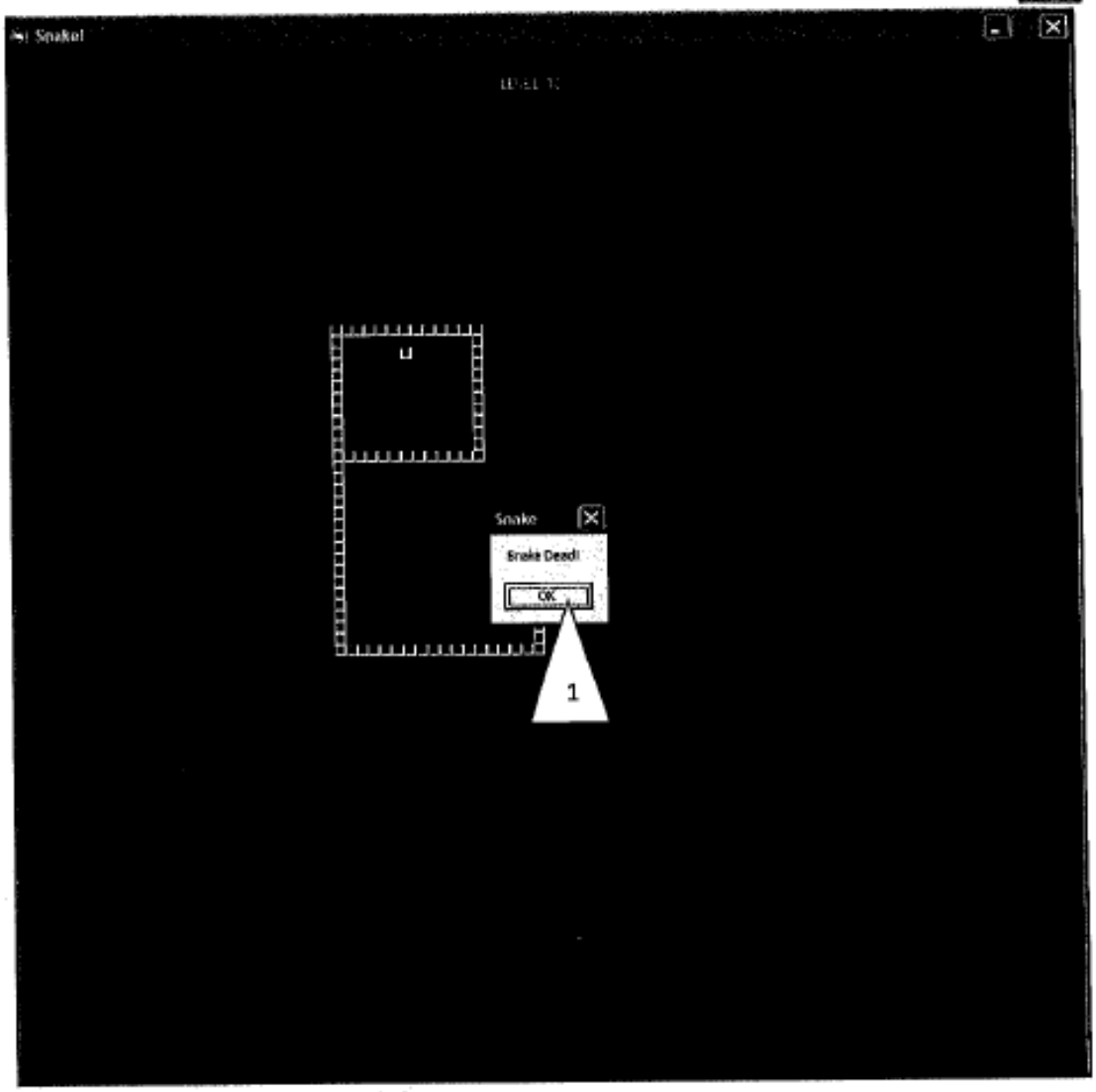
The game will end if the snake's head collides with its own body or with the borders of the game screen, as shown below. If this occurs, click the OK button (1) and you will be returned to the initial level selection screen.

This shows the snake colliding with border of game screen



13

This shows the snake colliding with its own body



4 – Troubleshooting

Snake is not a program that has error messages, and as such there are no error message issues to be covered here. There are a few issues that may arise, and I shall attempt to cover them and their solutions in the table below.

4.1 - Download and installation issues

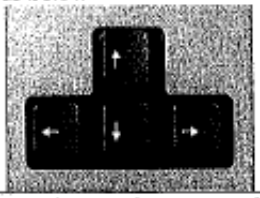
Issue	Possible cause	Solution
Unable to access website	1. You may not be connected to the internet	Contact your internet service provider for assistance
	2. The web site www.vbgamer.com may be having technical problems	Try accessing the website at a later time
Program will not save	You may have insufficient hard disk space	Ensure that you have enough hard drive space to accommodate the program. A minimum of 1 megabyte is required

4.2 – General issues

Issue	Possible cause	Solution
Program will not open	1. You may not have the appropriate operating system necessary to run the Snake game	Ensure that you install and run Snake on a computer with Microsoft Windows 95 or later operating system
	2. You may not have the Visual Basic 6 service pack that is necessary to run the Snake game	Download the service pack from the following website: www.support.microsoft.com/kb/290887
Snake appears to slow down when higher levels are reached	1. You may not have sufficient RAM	Ensure that you install and run Snake on a computer with at least 512 megabytes of RAM
	2. You may not have sufficient processing power	Ensure that you install and run Snake on a computer with at least 1 processor with a clock rate of 1.3 GHz or more
	3. If the above two causes do not apply, then this problem may be the result of a known issue with this version of Snake, that has no solution at present	NA

5 - Glossary

15

Term	Definition
Address bar	The horizontal bar displayed at the top of your web browser that, when selected, allows you to enter a web site address with the keyboard
Clock rate	The rate in cycles per second at which a computer performs its most basic operations – in general the greater the clock rate of the processor, the faster the computer can perform processing tasks such as those involved in the running of the Snake game
Directional arrows	The buttons on the keyboard that appear as four arrows as below 
Drop-down list	An object with an arrow that can be clicked on, showing a list with items that can be selected
Form	A screen used for viewing and/or inputting data
GHz	Short for gigahertz. It is unit used in measurement of frequency. One GHz is equivalent to one billion cycles per second
Hard disk	A storage device used to store data on your computer
Level	A number that indicates your progress through the game. Level numbers are incremented every time you direct the snake onto five food items. The higher the level number, the greater the difficulty of the game
Megabytes	A unit used in measurement of data. Is roughly equivalent to one million bytes
Operating system	The software that enables your computer to run programs, such as Snake. Snake is only compatible with Windows operating systems
Processor	The component of your computer that carries out the instructions of programs that you run, such as the Snake program
RAM	Short for random access memory. It is the memory of your computer that is used to store data for processing by the processor. In general, an increase in the amount of RAM in a computer leads to an increase in the computer's performance
Snake bits	Snake bits are the individual green blocks that make up the snake's head and body
Visual basic service pack	Refers to the files that are required by all applications that are created with Visual Basic 6.0, such as Snake
Web browser	A program used to browse web sites on the internet

Section 5 – Evaluation

Part 1 – The degree of success in meeting the original objectives, and an evaluation of the project's management

After having completed the system, I can conclude that all but one of the requirements decided upon in the refined design objectives on page 27 have been met. The requirement that has not been met is the output requirement 4.b, which is that: "The positions of each snake bit will be displayed in real time". I refer to the requirements as those laid in these refined design objectives, as opposed to those in the original requirements specification on page 14, because the design objectives are essentially the same as the requirements of the specification, except they contain additional requirements that were decided upon as a result of further user consultation.

The requirement 4.b has not been met due to a lack of foresight when designing the way in which the snake was to move. It never occurred to me, when designing the algorithms, that the program could cause a modern computer any difficulty in its execution. Thus, I did not seek to find the most efficient solution to the problem of moving the snake, but simply a solution that would work.

My users, after having accepted the program's shortcomings in failing to meet this one requirement, unanimously agreed that the program succeeds in meeting all other requirements. This can be seen from their questionnaire responses to beta testing, outlined on page 88.

I shall now discuss each objective of the requirements specification and explain why each of them, apart from 4.b, was successfully met.

Requirement	Explanation of why the requirement was met
1.a) The screen size is large – (900x900 pixels)	This requirement was met because the screen size of the snake game screen was, as planned, 900x900 pixels. All seven of my users can be seen to have testified to this fact in their response to question 12 of the user testing questionnaire on page 88
1.b) The colour of the snake will be: green	The colour of the snake was indeed green, as can be seen from many of the screen shots from page 58 onwards
1.c) The colour of the food will be: pink	The colour of the food was indeed pink, as can be seen from many of the screen shots from page 58 onwards
1.d) The colour of the background will be: black	The colour of the background was indeed black, as can be seen from many of the screen shots from page 58 onwards
1.e) The colour of the background for the menu screens will be: dark grey	The colour of the background was indeed dark grey, as can be seen from the screen shots 47 and 48 on pages 82 and 83
1.f) Each 'bit' of the snake will be 10x10 pixels in size	Each snake bit was 10x10 pixels in size, as agreed with the users. The same sized objects that I showed my users (as seen on page 10) were then used in the final program
1.g) The snake should start at a length of thirty 'bits' on level 1	This can be seen to be true from the result of level selection testing, as evidenced in screenshot 1 on page 88
1.h) The snake's head should initially be facing right	This was true as the snake always moved right as a new game began.

1.i) Each food item will be 10x10 pixels in size	Each snake bit was 10x10 pixels in size, as agreed with the users. The same sized objects that I showed my users (as seen on page 10) were then used in the final program
1.j) The time interval between each snake movement at level 1 will be 100 milliseconds	This requirement was met, as is seen in screenshot 44 on page 79 that demonstrates the timer interval on level 1
1.k) There is no pre-defined limit on the number of levels to be played through (instead this is limited by the skill of the player)	This requirement was met, as can be seen as a result of level advancement testing, as evidenced in screenshot 46 on page 81
2.a) The player is able to choose the level at which he/she begins the game from (between 1 and 10) with the use of a drop down list and combo box on both the initial menu screen and the instructions screen	This requirement was met, as can be seen from the level selection testing on page 51, the results of which are demonstrated in screenshots 1 to 10 from page 58 onwards
2.b) The player is able to control the direction that the head is facing with the use of the directional keys on a keyboard	This requirement was met, as can be seen from the movement of snake testing on page 52, the results of which are demonstrated in screenshots 11 to 22 from page 63 onwards
2.c) The player will not be able to make the snake go back on itself by pressing left when the head is facing right, or up when the head is facing down etc.	This requirement was met, as can be seen from the movement of snake testing on page 52, the results of which are demonstrated in screenshots 11 to 22 from page 63 onwards
2.d) The player will be able to view an instructions screen that opens in a new form by clicking a command button that is on the initial menu screen	This requirement was met, as can be seen from the instructions display testing on p57, the results of which are displayed in screenshot 47 on page 82
2.e) The player will be able to go back to the initial level selection screen from the instructions screen with the use of a command button	This requirement was met, as can be seen from the returning to initial menu screen from the instructions screen testing on p57, the results of which are displayed in screenshot 48 on page 83
3.a) The coordinates of the snake's head need to be continually incremented by 10 pixels in the direction that the snake's head is facing at a time interval that is determined by the level chosen	This requirement was met, as can be seen from the movement of snake testing on page 54, the results of which are demonstrated in screenshot 28 on page 71
3.b) Each snake bit behind the head must be incremented by 10 pixels in the direction that the snake's head was facing on a previous move (with the move number corresponding to the snake bit's distance from the head, such that the snake's body will follow the path of the head)	This requirement was met, as can be seen from the movement of snake testing on page 54, the results of which are demonstrated in screenshots 23 to 28 from page 69 onwards

3.c) It must be determined with each movement whether the snake's head collides into its body, in which case the game should end	This requirement was met, as can be seen from the collision with snake detection testing on page 55, the results of which are demonstrated in screenshots 38 to 39 from page 76 onwards
3.d) It must be determined with each movement whether the snake's head collides into the borders of the game screen, in which case the game should end	This requirement was met, as can be seen from the collision with walls detection testing on page 54, the results of which are demonstrated in screenshots 29 to 37 from page 72 onwards
3.e) It must be determined with each movement whether the snake's head collides into a food item, in which case the a the food item will be randomly moved to another position on the game screen	This requirement was met, as can be seen from the collision with food detection testing (with no level advancement) on page 55, the results of which are demonstrated in screenshots 40 to 41 from page 77 onwards
3.f) The system must record the amount of food items eaten, and increment the level number every time 5 food items are eaten	This requirement was met, as can be seen from the collision with food detection testing (with level advancement) on page 56, the results of which are demonstrated in screenshots 42 to 43 from page 78 onwards
3.g) The time intervals between each movement of the snake must be decreased by 10 milliseconds every time the level number is increased	This requirement was met, as can be seen from the collision with food detection testing (with level advancement) on page 56, the results of which are demonstrated in screenshots 44 to 45 from page 79 onwards
3.h) At each new level the length of the snake must be increased by 6 bits	This requirement was met, as can be seen from the collision with food detection testing (with level advancement) on page 56, the results of which are demonstrated in screenshots 42 to 43 from page 78 onwards
4.a) The current level number will be displayed	This requirement was met, as can be seen in the many in-game screenshots displayed on page 58 onwards
4.b) The positions of each snake bit will be displayed in real time	This objective was not fully met, due to the inefficiency of the algorithm that controls snake movement. The poor performance of the program when run was revealed too late in the development process to implement a new method of snake movement, as it would have led to a complete re-design of the algorithms controlling snake movement and those controlling snake length. Instead, a partial fix of the problem was implemented, by incorporating the CheckSnakeCollision subroutine into the MoveSnakeBody subroutine, whilst swapping the i's and k's in the snake collision detection algorithm around so that there were less loops being executed by the program. The method of implementing a complete fix to this issue will be discussed in the final section of the project concerning desirable extensions.
4.c) The position of the food item will be displayed	This requirement was met, as can be seen in the many in-game screenshots displayed on page 58 onwards
4.d) Instructions on how to select a level be displayed on the initial level selection screen	This requirement was met, as can be seen from screenshot 48, on page 83
4.e) Instructions on how to play the game will be displayed in a new form when the instructions command button is clicked	This requirement was met, as can be seen from the instructions display testing on p57, the results of which are displayed in screenshot 47 on page 82

Part 2 – An evaluation of the user's response to the system

In order to clarify the extent of satisfaction that my user group has with regards to the finished program, I have designed a new questionnaire intended to complement the questionnaire completed by my users in the user testing section on page 87, and fully gauge the level to which the final programme is accepted by my users and therefore the extent to which the program is 'user-friendly'. The questionnaire, and the frequency of each of my users' responses as indicated in parentheses, are as follows.

Question	Answer (Y/N)	Comment
Were the menus clear?	Y (7) N (0)	None
Was the on screen help clear?	Y (7) N (0)	None
Could the system be easily used?	Y (7) N (0)	None
Are you satisfied with the completed program as a fulfilment of the design requirements, despite the fault with the snake's movement that becomes apparent at the higher levels?	Y (7) N (0)	None
These are the issues highlighted by user testing that I sought to resolve: 1) The player should be informed that there are more levels beyond level 10 2) It should say at the beginning of the instructions screen how you are to control the snake 3) The instructions form appears at a different place on the screen from the initial menu screen – it should instead appear in the same place Are you satisfied that these issues have been fully resolved?	Y (7) N (0)	None
Are there any other issues that you feel remain to be fixed?	Y (0) N (7)	None

In both this and the questionnaire on page 87, the users' responses to the completed program have been very positive. A few issues were initially highlighted by my users, but they are now satisfied that these issues have been resolved, and they have each agreed that the system adequately fulfils the design requirements- despite the fault with the snake's movement that becomes apparent at the higher levels. Therefore this fault remains only as a consideration in any desirable extensions that could be made to the system in future. This shall be discussed in the final part of the project.

Part 3 - Desirable extensions

In evaluation of the system, I shall list the good and bad points of the completed program and note its limitations, and how these may be resolved with possible extensions.

Good points

- As concluded from the review of my users feedback from the questionnaire on the previous page, the system can be deemed 'user-friendly'
- Apart from the slow-down that hinders the later levels, the program functions as desired in the initial requirements specification – as testified to by my users in the questionnaire on the previous page and the user testing feedback presented on page 87
- The look of the completed system closely matched that of the designs, and was a subject of positive response from my users
- The instructions were deemed by my users to be clear
- No bugs exist in the system, according to my alpha testing on page 37 and my users' beta testing
- The game had an appropriate difficulty (was neither too difficult nor too easy), as stated by my users in their feedback
- All buttons/combo boxes work as desired

Bad points

- The only limitation of the program is its inability to maintain a constant snake speed as the level number increases above 15, due to the inefficiency of the code controlling snake movement. This could be resolved in a new version of the program, where the snake's movement and all related functions are controlled in a different way

Possible extensions

The extension that would need to be implemented in a later iteration of the program is a new, more efficient method of moving the snake bits across the screen. In the present version, the program loops through and positions every individual snake bit in turn, when a far more efficient method that would have produced the same results would have been to simply move the head and final snake bit of the snake.

As the program already moves the snake's head independently of the following snake bits, a good method of changing the algorithm controlling snake movement would be to move only the final snake bit in the CheckSnakeCollisionAndMoveSnakeBody subroutine, and to move it to the space that is left behind the snake's head when that has been moved. This would drastically decrease the amount of loops carried out by the program, and I believe that this would remove the present issue the snake appearing to slow down when it gets very long.