DEFINITION, INVESTIGATION AND ANALYSIS

(i) PROBLEM DEFINITION

Bejeweled is a famous puzzle game developed by Pop Cap Games, a company formed in 2000. Initially named Diamond Mine and used to be a Flash game, Bejeweled has now become one of the most popular games on PC as well as on mobile phones. Pop Cap has released three versions of Bejeweled, with the latest, Bejeweled 3 released in December 2010, featuring high quality graphics.

Bejeweled is an example of a Connect-3 game, where the users are allowed to swap an item (e.g. a gem in Bejeweled) with an adjacent item by clicking on them.

The objective of the game is to form horizontal or vertical chains of three or more similar items (e.g. blue gems). These chains will disappear and the player is awarded with points. Bonus points are given when the chain is more than three items or two chains are created just in one swap. Items will then fall from the top to fill the gap. When the player has gained a certain number of points he/she will go to the next level, which is more difficult.

In these Connect-3 games, the game ends when the time bar has reached the end (the player has run out of time) or when the player cannot make any more possible move.

There are also many other versions of Connect-3 games, which differ in features such as the graphics- items can be diamonds, sushi, balls, etc. The rules of the games are also slightly different.

By working with Visual Basic 6.0, I am developing a simple version of a Connect-3 game which aims at users of all ages since the game is entertaining and suitable for everybody.

Here I include screen shots from some versions of Connect-3 games and the main features of these:

This is the original *Diamond Mine* produced by Pop Cap. This version allows users to choose between two game modes, Normal and Time Trial. The game clearly shows the current points and the points recently gained.

The visual effect of items falling is clearly shown. Time limit is represented by a time bar.

This is the latest Bejeweled 3 by Pop Cap, featuring high-definition graphics:

There are 4 different game modes to choose from: Classic, Zen, Lightning and Quest.

4 other secret game modes are locked at the beginning (small buttons with question marks), each of which will be unlocked when the user successfully completes a certain level of that main mode.

The player can choose a theme (*Ice Storm* in the picture below) and the background will represent the theme. The objects are transparent (grid, time bar) so we can still see the background behind these objects.

The game clearly shows the current points and the points recently gained. Each chain of 3 created is awarded with 50 points. More points are awarded for chain of 4, 5, and combos.

The visual effect of items falling is clearly shown. Time limit is represented by an animated time bar.

Some Connect-3 games have a plot/story behind them. For example this game below is a battle between two characters (Pixel and Python in this particular game) played by 2 users, or by a single user and the computer. The objects represent different weapons that one can use to attack the other. When the user creates a chain of 3, an attack has been made and the opponent's life and energy level has been reduced. When Pixel or Python runs out of life (green bar) or energy (yellow bar), it loses and the other wins.

(ii) INVESTIGATION AND ANALYSIS

In order to understand the requirements of the users for the game, I am going to carry out an investigation by doing a few interviews with my users.

The first interview will be paper-based (my users will fill in prepared forms) and will cover general points about the game, such as how the game screens should look like, what the theme should be, how the scoring system works, etc.

After doing the first interview and collecting the answers of users, I shall summarise the result and produce a draft requirements specification based on that summary, also taking into account the scale of the project. This draft specification will then be proposed to my potential users. If the majority of users agree with this specification, I will ask them further questions to enable me to build a complete specification.

FIRST INTERVIEW:

I am going to interview some students aged between 16 and 18, from various sixth form colleges. I shall include next page a sample of the form which my potential users are going to fill in:

As a part of my Computing A-level coursework, I am developing a simple version of a Connect-3 game with Visual Basic 6. This game involves swapping adjacent items in a grid to create a chain of 3 or more identical items and the player will be awarded with points for creating these chains.

As you are a potential user of the game, please take a moment to complete the interview. Your opinions will help me build a complete requirement specification for the programme.

1. **Have you played a version of a Connect-3 game before (e.g. Bejeweled)? If yes, what do you think was most enjoyable about the game?**

2. **At the starting point, would you prefer a small grid (e.g. 5x5), medium (e.g.8x8), or large (e.g. 10x10)?**

3. **How many different items should be in the game at the starting point?**

a. 3

b. 4

c. 5

4. **Would you prefer a timed or untimed game?**

5. **Would you prefer the game to have different levels of difficulty?**

(If yes please consider the next few more questions below, otherwise please move on to question 6)

When the player is on a new level, some features should be changed to increase the difficulty of the game. Please tick in the box(es) if you think any of these changes are necessary:

Increase in size of grid	
Increase in number of different items	
Decrease in time limit	

6.

	Yes	No	Added comments
Should the game allow swapping several times to make a chain?			
Would you prefer a timed game?			
Should the game only allow swapping vertically and horizontally?			
Should the game allow chains of more than 3 items?			

7. What is the scoring method that you like? I.e. should the players be awarded more points when they create a chain of more than 3/ create 2 chains in only 1 swap?

8. When a chain of 3 or more is created, the items in that chain can vanish immediately or gradually before new items drop down from the top. Please state how you would like the chain to vanish:

a. A small “explosion” clears out the chain(s)
b. Smiley faces replace all the items involved in the chain before new items drop down
c. Involved items all disappear at once and new items gradually drop down

9. What should be the theme of the pictures (items)?

a. Diamonds
b. Food (e.g. Sushi)
c. Fruit
d. Animals

Please freely use this space below to add any more details about the pictures (items) that you wish the game to include (e.g. if you choose fruit as the theme, you can add “bananas, apples, oranges, kiwi”)

10. Should the game allow players to choose a theme?

11. How should the background look like? What is your colour preference for the background?

SUMMARY OF THE FIRST INTERVIEW

Included here is my summary of the results of the interview, in the form of tables. The order of the summary is not similar to that of the interview, however presenting clearly all popular answers and added comments of users.

The numbers in brackets indicate the number of users who choose a particular answer.

* For questions about the theme (question 8) and how to clear out chains (question7) and what features should be changed when a new level is reached (question 4), users can choose more than 1 answer; therefore the number in brackets in each column may not add up to 7.

Popularity	Theme	Background	Number of items at start	Size of grid at start	Clear out chain
1	Diamonds (4)	Plain, single colour (black/brown/light), not too colourful (5)	Four (3)	Small (3)	Smiley faces(3)
2	Animals (3)	Background that fits with the items e.g. for sea creatures the background should be ocean. (2)	Three(2)	Medium(2)	Explosion(3)
3	Fruits (1)		Five (2)	Large (2)	Disappear at once- items drop gradually (1)
Added comments of users	-Rare gens: diamonds, rubies, sapphires, emeralds, topaz… - Animals: lions, elephants, snakes, wolves, giraffes… -Animals: myth creatures -Fruit: fruit ninja (bananas, water melon…)	- The time bar/clock should be designed to make players feel frustrated in the hard levels (e.g. a clown's face)			

Questions	Yes	No	Added comments
Have you played a Connect-3 game before? What was most enjoyable about the game?	6	1	Most users answer that the challenge of playing against the time and collecting as many points as possible were most enjoyable.
Would you prefer the game to have different levels of difficulty? What feature(s) should be changed when a new level is reached?	7	0	<u>Features should be changed</u>: -Decrease in time limit (6) -Increase in number of items (6) -Increase in size of grid (4)
Would you prefer a timed game	6	1	Some users added that a timed game would make the game much more fun and challenging.
Should the game allow swapping several times to make a chain?	1	6	Most users said it would be too easy if the game allows several swaps to make a chain. A swap should only be allowed if a chain is made afterwards.
Should the game only allow swapping horizontally and vertically	6	1	
Should the game allow chains of more than 3?	6	1	Some users added that they want bonus points if they make a chain of 4/5 or 2 chains in 1 swap e.g. -for a combo : x number of items in the chain) -for multiple chains: x2 per chain
Should the game allow players to choose a theme?	5	2	- A user wrote that the game would be more attractive to players if it allows players to choose a theme.

CONCLUSIONS FROM THE FIRST INTERVIEW:

1. Most users said what they enjoy most about a Connect-3 game is the challenge of making as many chains as possible in a limited amount of time to be able to get on new levels as well as achieving highest scores possible. The game, in their opinion, should be timed and should have different levels of difficulty.

2. Most users prefer a small grid at the beginning and the size of grid can increase as we go to higher levels.

3. The most voted number of different items is 4 and users would like to see this number increasing in higher levels.

4. Swapping adjacent items should only be made when a chain is created and only vertical and horizontal swaps can be made.

5. The users would like to see the scores going up more when they create a chain of more than 3 items and combos.

6. Most users would like a theme of Diamonds or Animals for the items. Some indicated jungle/ocean animals.

7. Players can choose a theme and the background should therefore reflect the theme of items, e.g. a theme of jungle animals should have background of jungles. Some users on the other hand preferred a plain and simple background such as brown/black.

DRAFT SPECIFICATION

Based on this summary and also taking into account the scale of the project, I am going to propose a draft specification to my potential users. This draft specification includes basic features of the game. If the majority of users agree with this specification, I will ask them further questions to enable me to build a complete specification.

I am developing a Connect-3 game named “Happy Ocean” .The goal of the game is to score as many points as possible by creating chains of 3 (or more) identical items, in a limited amount of time and without using more than 2 swaps. The basic features will be:

- A Menu page showing 3 options for users: Instructions, Start Game and Quit.
- An instruction screen will show up when the button ‘Instructions’ is clicked on.
- When the user clicks on the ‘Quit Game’ button, he/she will be prompted with a message ‘Are you sure you want to quit?’ .If ‘Yes’ is clicked, all screens will be unloaded. If ‘No, not yet’ is clicked, the prompt message will be closed, giving back the Menu screen.
- Clicking ‘Start Game’ will lead to Gameplay screen. The interface of this screen includes a grid of size 8x8, a score box, a counter that keeps track of the number of swaps available, and a timer.
- When the button ‘Play’ is clicked, pictures will be loaded onto the boxes in the grid. These pictures are not yet decided.
- The user can swap an item with its adjacent item vertically and horizontally to create a chain of 3 or more identical items. Any invalid swap would lead to a ‘bleep’ sound.
- The game will allow more than 1 swap to create a chain, and the maximum number of swaps which can be made is 2. There will be a counter that keeps track of this number. When a swap is made and there is no chain detected afterwards, the number in the counter will be decremented.
- If the user creates a chain of 3 within this limit, the number in the counter will automatically become 2. If the number in the counter is 0 (which means the user has already used up all 2 swaps) and no chain is detected, then the game ends. A message box will pop up, saying “Sorry, you have used up 2 swaps without creating any chain!’. A Game Over screen will appear.
- Only one chain will be detected at a time. ‘Smiley faces’ will replace the involved items- this means that the chain has been removed.
- Items above the chain will drop down. The game will check again for any newly created chain, if there is then the process is repeated. Further checks will be carried out until no more chain is detected.

- Each time a chain is detected a certain number of points will be awarded. I suggest that a chain of 3 would score 30 points while a longer chain would score 50 points.

- There is also a time limit for the game. I suggest the game will be 5 minutes long and when time has run out the game ends. A message box will appear, saying “TIME’S UP”, leading to the Game Over screen.
- In the Game Over screen, the final score is showed to the player as well as 3 options: Try Again, Back to Menu and Quit Game.

Most of my users agreed with this draft specification, hence I shall ask them further questions, in the same format as the first interview, to make the specification more complete:

1) Would you prefer power-ups in the game? (A power-up appears inside the grid as an item; just by one click on that item the player can remove lots of other items and gain points without having to swap anything).

2) I have suggested that the time limit is 5 minutes. How long do you think the game should last?
3) I suggest that a chain of 3 would score 30 points while a longer chain would score 50 points. This scoring system will consequently make the final score relatively high (more than 1000).Please give your opinion about this scoring system- would you prefer high final scores (more than 1000) or lower scores (100-200)?

4) I have decided to go for the theme ocean animals/ creatures. Since the theme is ocean, there will be some pictures of ocean creatures on the Menu screen and the background colour is likely to be blue. Do you agree with this interface? Should the pictures on the Menu screen be different from the pictures in the grid (Play Game screen)?

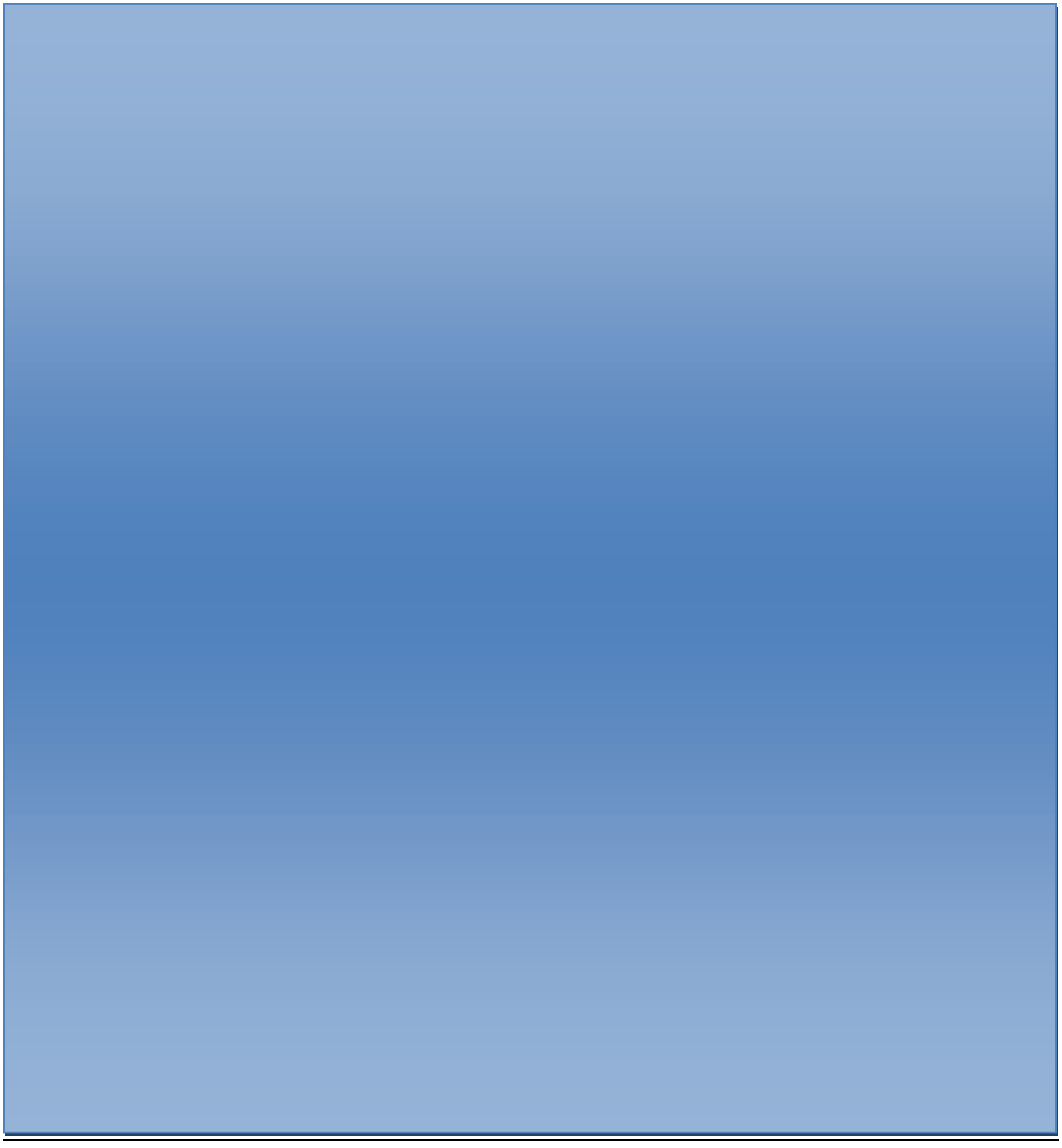
5) Please list 5 creatures whose pictures should appear in the grid, e.g. shark, whale, golden fish.

6) I suggest that the screen size is medium, i.e. not full screen. The main screen (Play Game) should be biggest, approximately 800x700 (demonstrated next page). Are you satisfied with this screen size?

7) What would be the background colour(s) for the Menu screen, Gameplay screen and Instructions screen?

This is my suggestion of the size of the Play Game screen:

(Page orientation: landscape)



SUMMARY OF THE SECOND INTERVIEW:

(1) Yes (3)
No (4)
Comment(s): adding power-ups can make the game too easy.

(2) Most of the users said 5-minute limit is too much for a simple game. They agreed 1 or 2 minutes would be more suitable

(3) High scores (5), low scores (2)

(4) Agree (7)
Pictures should be different in two different forms (7)
Comment(s): Using the same pictures will make the interface less attractive to users.

(5) Popular answers: starfish, octopus, crab, fish (general), prawn, whale, dolphin.

(6) Yes (7)

(7) For Menu screen:
Blue (7)
For Play Game screen:
Blue (5)
Dark grey (1)
Light green (1)
Extra comment(s): The two blue colours for two different screens should be different from each other, e.g. light blue for one screen and dark blue for another.
For Instruction screen:
Sandy yellow (4)
Blue (2)
Violet (1)

After doing two interviews to get all the users' opinions and careful consideration of the time span and scale of the project, I am now be able to produce a complete user requirements of the game.

USER REQUIREMENTS:

- Design requirements:

1) The screen size is medium (i.e. not full screen) and varies for different forms. The Play Game screen must be largest among all (approximately 800x 700)

`

1) The background colour for the Menu and Play Game screen will be blue, some pictures of ocean creatures will be included in the screen (e.g. at the corners)

2) Initial selection screen (Menu) includes options 'Start Game', 'Quit Game' and 'Instructions'

2) 'Start Game' button leads to the gameplay screen. This gameplay screen has an 8x8 grid, each box is 60x 60 pixels

3) The 5 pictures placed in boxes of the grid are:

   Starfish, fish (general), dolphin, seahorse, crab

4) There is a timer starting at 120 (seconds) and a score box on the left hand side of the grid

5) There is a counter that keeps track of the number of swaps available on the left hand side of the grid.

- Input requirements:

1) The player can click on the 'Instructions' button on the Menu screen to view the instructions

2) The player can go back to Menu screen after reading the instructions.

3) The player can go straight to a new game after reading the instructions by clicking 'Start Game' on the Instructions screen.

4) The player can click on button 'Start Game' on Menu screen to start a new game

5) The player can click on button 'Quit Game' on Menu screen to exit the game. This will lead to a prompt message, asking the player 'Are you sure you want to quit the game?'

6) On the prompt message screen, the player can exit the game by clicking 'Yes' and stay in the game by clicking 'No, not yet'

7) The player uses the left mouse button to click on two items to swap them

8) The player can click on 'Try Again' on the Game Over screen to start a game.

9) The player can click on 'Quit Game' on the Game Over screen to exit the game. This will again lead to a prompt message.

10) The player can click on 'Back to Menu' on the Game Over screen to go back to Menu screen.

- Processing requirements:

1) The player can only swap a picture with an **adjacent** picture **vertically** or **horizontally,** by clicking on the two boxs**.** Each time any two boxes are clicked, the system has to be checked if that is a valid swap. An invalid swap would lead to a 'bleep' sound.

2) There are maximum 2 swaps which the player can make to create a chain. A counter will keep track of this number.

3) It must be determined with each swap whether the number in the counter is 0 (which means the player has already used up all 2 swaps) while no chain is detected. In that case the game ends, with a message box saying 'Sorry, you have used up two swaps without creating any chain!'

   Clicking button 'OK' on this message box will lead to the Game Over screen.

4) After each swap, the game will check whether a chain of 3 has been created. **Only the first chain** found in the checking process will be detected.

5) The number in the counter will be set back to 2 after a chain has been detected.
6) Award 30 points for any chain of 3 and 50 points for any longer chain. The system must record the points awarded and output the new score.

7) The items in the detected chain will be replaced by pictures of a yellow smiley face (demonstrated below)

8) Items above the chain will drop down. The top boxes will be filled in with new pictures, which are randomly chosen.

9) The checking process is carried out again to detect if any new chains were formed by the previous "remove and drop down" process. The checking process only stops when no more chains are detected.

10) When the timer has come to 0 (it started counting down from 120 seconds) the game should end immediately, showing a message box "TIME'S UP".

Clicking button 'OK' on this message box will lead to the Game Over screen.

- Output requirements:

1) Display instructions on how to play the game when the user clicks on the 'Instruction' button
2) Display time and current scores
3) Display final score and options 'Try Again', 'Quit game', 'Back to Menu' in the Game Over screen.

- Hardware and software requirements:

Software:

- Any operating system among Microsoft Windows 95, 97, 98, 99, 2000, XP, Vista, and Windows7

- Visual Basic 6 DLLs

Hardware requirements:

- Processor:

-Ram 32MB

-10 MB of hard-disk space

- VGA

- Computer mouse

- User review of requirements:

All of my users have reviewed this specification and agreed to the requirements listed.

Signature (with name printed below):

DESIGN AND NATURE OF SOLUTION

(i) NATURE OF THE SOLUTION

AIMS AND OBJECTIVES

MY GOAL IS TO CREATE A GAME…

with this interface:

- Screen size: medium i.e. not full screen. The main screen (Play Game) should be approximately 800x700 pixels
- Background colour for Menu and Play Game screen is blue
- Background colour for Instructions screen is yellow (sand yellow)
- 5 items will be: fish(general), starfish, crab, seahorse, dolphin
- An 8x8 grid
- Size of each box: 60x60
- A score box showing current score
- A timer starting at 120 (seconds)
- 5 different pictures representing 5 types of items that will appear throughout the game. These 5 pictures are set invisible to users
- Name of game at the top
- Yellow smiley faces replacing a chain of three

The game will perform these functions:

- Shows instructions if the command box “Instruction” is clicked on
- Allows users to go back to Menu screen from Instructions
- Allows users to swap two adjacent items horizontally or vertically by clicking on them
- Results in a “bleep” sound when an invalid swap occurs (e.g. cross swap)
- Checks for a chain of three or more identical items after each move
- Allows maximum 2 swaps to create a chain
- Keeps track of the number of swaps used by a counter
- Detects the first chain found
- Replaces the chain with smiley faces
- Removes the chain
- Increase 30 points for a chain of 3 and 50 points for a longer chain
- Sets number of swaps to 2

- Shifts upper items downwards (the visual effect is not included in the requirement)
- Creates new items at the top
- Checks for chain again
- Sets time limit of 120 seconds
- Output message box “TIME’S UP” when time has run out
- Output message box “Sorry, you have used up two swaps without creating any chain!” when no chain is detected after 2 swaps have been made
- Shows the Game Over screen when button ‘OK’ on these two message boxes above being clicked

The game will require this hardware:

- Processor:

-Ram 32MB

-10 MB of hard-disk space

- VGA

- Computer mouse

The game will require this software:

- Any operating system among Microsoft Windows 95, 97, 98, 99, 2000, XP, Vista, and Windows7

- Visual Basic 6 DLLs

User review:

INTERFACE DESIGN

Menu Screen

Object	Property	Setting
frmMenu	Caption	Menu
	Border Style	1-Fixed Single
lblMenu	Caption	Happy Ocean
cmdStart	Caption	Start Game
cmdInstruct	Caption	Instructions
cmdQuit	Caption	Quit Game
imgWhale1, imgWhale2, imgFish1, imgFish2, imgStarfish, imgCrab	Picture	Bitmap
	Stretch	True
	Appearance	1-3D

The sketch of this form is demonstrated on a separate sheet which is attached to this document, right after this page.

Instructions screen

Object	Property	Setting
frmInstruction	Back Colour	&H00C0FFFF& (bright yellow)
	Caption	Instructions
cmdStart	Caption	START GAME
	Font	Lucida Handwriting, size 12
cmdBack	Caption	BACK TO MENU
	Font	Lucida Handwriting, size 14
lblInstruction1	Font	Franklin Gothic, size 12
	Caption	You can click on two adjacent items to swap them. You can only swap vertically or horizontally. An invalid swap will result in a "bleep" sound. Every time you create a chain of three identical items within two swaps, you will be awarded 30 points. If the chain is longer than 3, 50 points will be awarded. However if you exceed 2 swapping times and no chain is detected, then the game ends.
lblInstruction2	Font	Franklin Gothic, size 12
	Caption	The goal of the game is to create as many chains as possible in 120 seconds (2 minutes) and also without exceeding the number of swaps allowed. Chains created will be removed and items above the chain will drop down.

Similarly, the sketch of this form is demonstrated on a separate sheet which is attached to this document, right after this page.

Play-Game Screen

Object	Property	Setting
frmPlayGame	Caption	Happy Ocean
	BackColour	&H00C0C000& (green-blue)
lblTitle1	Caption	Swaps Available
lblCounter	Caption	2
picAnimal1, picAnimal2, picAnimal3, picAnimal4, picAnimal5	Picture	Bitmap
	Visible	False
picArray	Index	1 to 64
	Picture	None
lbl1Title3	Caption	Time left
lblCountdown	Caption	120
lblTitle2	Caption	Score
lblScore	Caption	0

Similarly to the Menu and Instructions screens, the sketch of this form is demonstrated on a separate sheet which is attached to this document, right after this page.

VARIABLE TABLE

Variable name	Type	Size	Description	Sample value	Validation
PictureArray (1 to 64)	Byte	1-255	Stores the picture code of each box in the grid	2	PictureArray(n) and possiblemoves(n) cannot be less than 1 and greater than 64
possiblemoves (1 to 4)	Byte	1-255	Stores the index of the potential secondly-clicked box which would make a valid swap	55	
Flag	Boolean	0(False)-1(True)	Used to flag up if a box has been clicked on	False	NA
Flag1	Boolean	0(False)-1(True)	Used to flag up if a horizontal chain has been found	True	NA
Flag2	Boolean	0(False)-1(True)	Used to flag up if a vertical chain has been found	True	NA
click1	Byte	1-255	Stores the index of the first box being clicked	33	Click1 and click2 cannot be less than 1 and greater than 64
click2	Byte	1-255	Stores the index of the second box being clicked	45	
hor_start	Byte	1-255	Stores the index of the starting box of a horizontal chain	9	hor_start and hor_end must be greater than 0 and less than 65. (hor_end-hor_start)>=2
hor_end	Byte	1-255	Stores the index of the ending box of a horizontal chain	12	
ver_start	Byte	1-255	Stores the index of the starting box of a vertical chain	1	ver_start and ver_end must be greater than 0 and less than 65. (ver_end-ver_start)>=16
ver_end	Byte	1-255	Stores the index of the ending box of a vertical chain	17	
grid	Byte	1-255	Stores the size value of the grid	8	NA

			(which is the length of a side of the square)		
score	Integer	0-9999	Stores the current score of the user	250	NA
FinalScore	Integer	0-9999	Stores the final score of the user (when the game ends)	1000	NA

(ii)ALGORITHMS AND TESTING

Developing the game is a big project which requires the developer to use stepwise refinement. I will therefore split the project into small modules and these modules can be split further into smaller sub-modules and so on.

Here I shall list all my modules/routines in a structured way:

- **Set up grid**
  - ❖ Create a control array of picture boxes
  - ❖ Randomly place pictures into those boxes
  - ❖ Set counter to 2 and time to 120

- **Clicking and Swapping 2 boxes**
  - ❖ Decrement counter and check if counter is 0. If it is, output form frmUseUp and record final score.

- **Detecting chains**
  - ❖ Detecting horizontally
  - ❖ Detecting vertically

- **Remove and drop down**
  - ❖ Removing horizontally
  - ❖ Remove vertically

- **Scoring**
  - ❖ Display current scores
  - ❖ Increase 30 points for any chain of 3 and 50 points for any chain of more than 3
- **Timing**
  - ❖ Set up timer (120 seconds)
  - ❖ Display message box “TIME’S UP” when time has run out

I shall show the modules in the form of an organisation chart on a separate sheet which is attached right after this page.

I will now use a combination of pseudo-code and flow charts to describe the algorithms used in these sub-routines (mentioned in each bullet point❖)

- **Set up grid:**

Make PicAnimal1, PicAnimal2, PicAnimal3, PicAnimal4, PicAnimal5 invisible
Disable Timer

❖ **Create a control array of picture boxes**

Create a picture box named Box. Copy Box 63 times to get a control array: Box(n), n=1 to 64

❖ **Randomly place pictures into those boxes**

Explanation of approach:

-Each picture is represented by a picture code (1, 2, 3, 4, or 5)

-Each of 64 boxs/boxes will be given a picture code. An array stores this number for each box.

-Randomly generate a number between 1 and 5 for each individual item of that array

- Attach 'real' pictures to the boxs according to the picture codes given.

Randomly generate a number for each box:

```
01  Randomize

02  For n=1 to 64
03        PictureArray (n) = the integer value of (a random number * 5) + 1
04   Next n
```

Dry run:

In this trace table I shall test 2 values of n.

Lines	N	PictureArray(n)	Comments
01			Randomize the function
02	1		
03		3	
04,02	2		
03		1	

Attach 'real' pictures

```
01 For n = 1 To 64
02      Select Case PictureArray(n)
03          Case 1
04              Picture1(n).Picture = Animal1.Picture
05          Case 2
06              Picture1(n).Picture = Animal2.Picture
07          Case 3
08              Picture1(n).Picture = Animal3.Picture
09          Case 4
10              Picture1(n). Picture= Animal4.Picture
11          Case 5
12              Picture1(n).Picture= Animal5.Picture
13      End Select
```

*Animal1, Animal2, Animal3, Animal4, Animal5 are the names of the 5 different pictures used in the game. These pictures will be made invisible:

In this dry run I use the values of PictureArray(n) from the previous dry run.

Lines	N	PictureArray(n)	Picture1(n).Picture	Comments
01	1			
02		3		
03, 04, 05, 06				
07				
08			Animal3.Picture	Attach picture Animal3 to box 1
09,10, 11,12,13				
01	2			
02		1		
03				
04			Animal1.Picture	Attach picture Animal1 to box 2
05,06,07,08.09, 10, 11,12,13				

- **Set counter to 2:**

  lblCounter= 2

  lblCountdown= 120

- **Clicking and swapping items:**

Explanation of approach:

- When any box is clicked on, the system will run a check to see whether this box is the firstly clicked or secondly clicked. This is done by using a Boolean value (Flag= True or False).

- If the box is firstly clicked (Flag=False) then the system records the Index of that box. A number of valid swapping positions are determined (for a middle box, there are 4 possible boxs the user can swap with- above, below, to the right, and to the left).

- Special case applies to boxs at the edges of the grid. To check whether a box is at the edge, we take Index1 divided by 8 (size of grid). If the remainder is 0 it means the box is at the right edge of grid, i.e. 3 swapping positions are above, below and to the left.

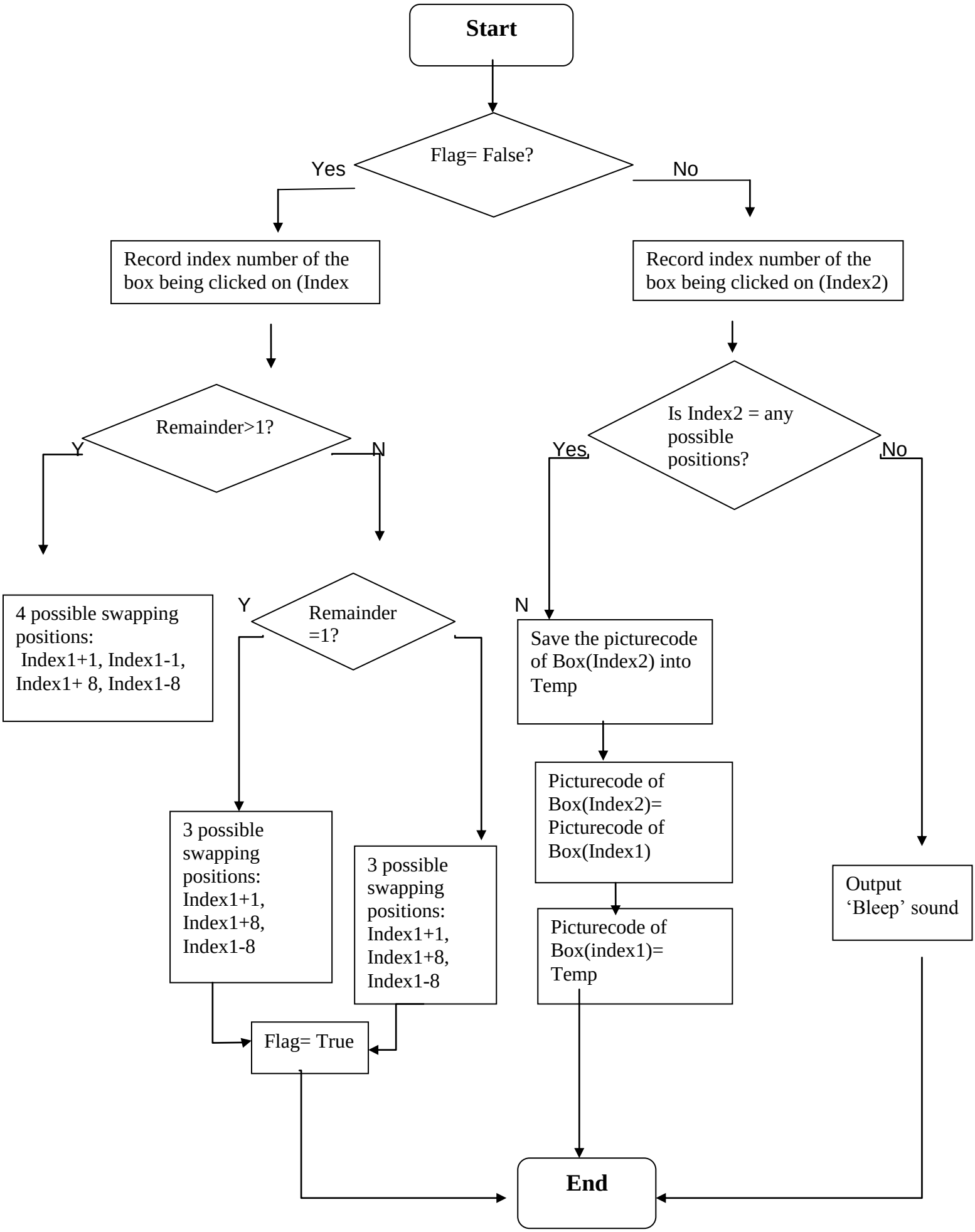
If the remainder is 1, the box is at the left edge, i.e. 3 positions are above, below and to the right.

Remainder= Index1 Mod 8

-The system will then flag up (Flag=True)

- If the box is secondly clicked (Flag=True), the Index of this box is also recorded (Index2). This index is compared with the 4 possible swapping positions, if it equals to any of these 4 then the swapping process starts. Flag is set back to False.

Algorithm showed by flow chart:


Start
Flag= False?
Yes
No
Record index number of the box being clicked on (Index
Record index number of the box being clicked on (Index2)
Remainder>1?
Y
N
Is Index2 = any possible positions?
Yes
No
4 possible swapping positions:
Index1+1, Index1-1, Index1+ 8, Index1-8
Y
Remainder =1?
N
Save the picturecode of Box(Index2) into Temp
Picturecode of Box(Index2)= Picturecode of Box(Index1)
Picturecode of Box(index1)= Temp
3 possible swapping positions: Index1+1, Index1+8, Index1-8
3 possible swapping positions: Index1+1, Index1+8, Index1-8
Output 'Bleep' sound
Flag= True
End

- ## Detecting a chain:

❖ **Detecting horizontally:**

Explanation notes for flow chart:

-2 pointers, *start* and *end,* keep track of the starting box and ending box of the chain.

- The system will loop from the 1st to 62nd box (outer loop)

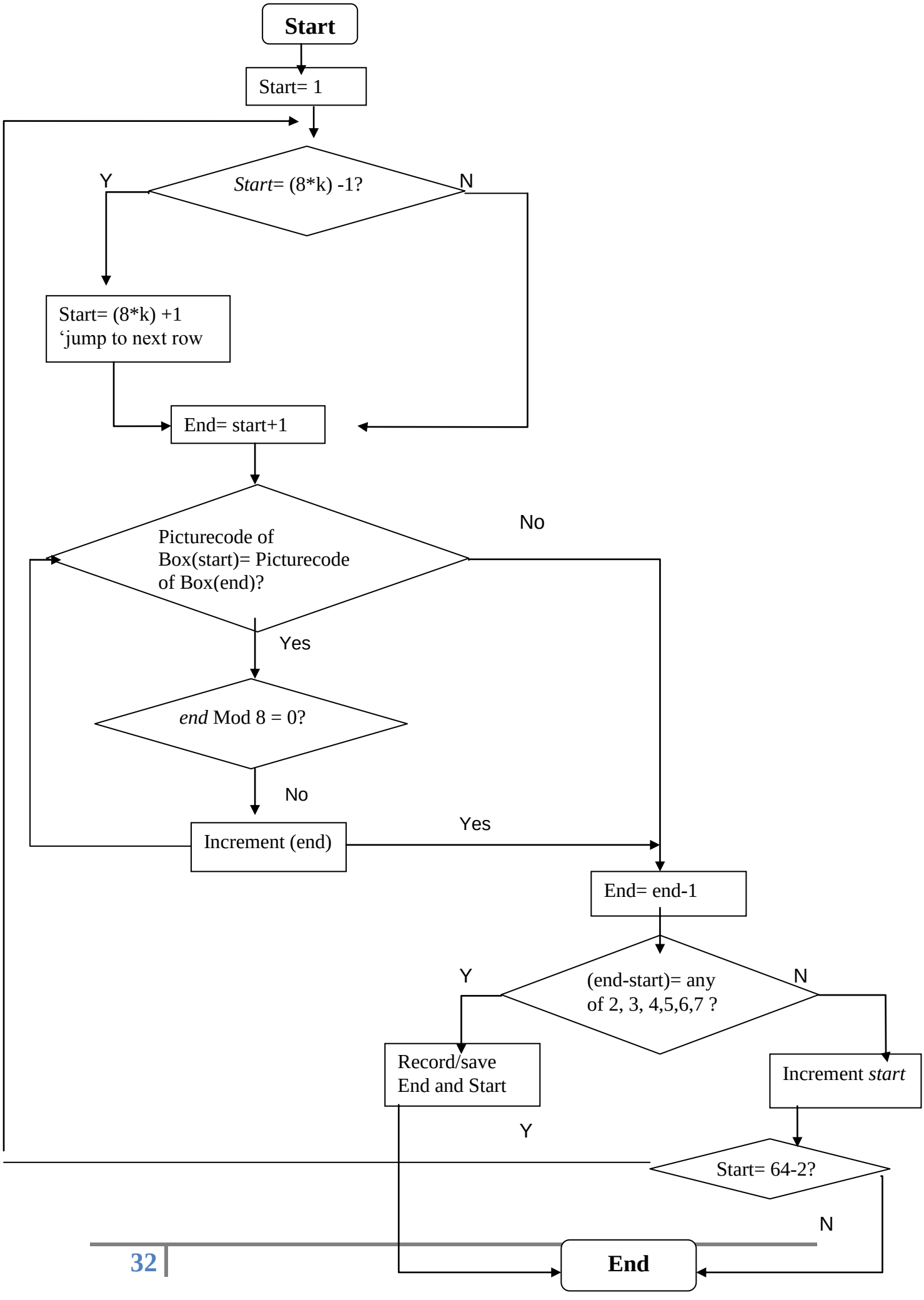
- If the box is near the right edge of grid (second to last box):

start= 8* (1 or 2 or or 4 or… 7) -1

start will become the first box of next row: *start*= 8*(1 or 2 or 3…or 7) +1

Notice: a variable k stores these values 1 to 7

- *end* is incremented every time picturecode of Box(end)= picturecode of Box(start). However if *end* is already at the right edge of the grid: **end Mod 8 =0,** the comparing process will stop.


Start
Start= 1
Y
Start= (8*k) -1?
N
Start= (8*k) +1
'jump to next row
End= start+1
Picturecode of Box(start)= Picturecode of Box(end)?
No
Yes
end Mod 8 = 0?
No
Increment (end)
Yes
End= end-1
Y
(end-start)= any of 2, 3, 4,5,6,7 ?
N
Record/save End and Start
Increment start
Y
Start= 64-2?
N
End

- **Detecting vertically:**

Explanation of approach:

- Similarly to detecting horizontally, the system will loop through the grid, but only from 1st box to 48th box, because when s*tart* reaches the 2nd to last row there can't be any more vertical chains.

- When *end* reaches the last row (*end*>=57) then *end* shouldn't be increased by 8. The comparing loop should end.

Pseudo-code:

```
For start= 1 To 48

    end = start + 8

    While l = 0 And PictureCode(start) = PictureArray(end)

        If end < 57 Then

            end = end + 8
        Else
            l = 1
        End If

    Wend

If l = 0 Then
        end= end - 8

End If

    If (end -start) >= 16 Then

        flag = True

        chainstart = start

        chainend = end


        start= 62
```

```
            End If

Next n

    If flag2 = True Then

        Call Remove_vertical(chainstart1, endchain1)
   End If
```

- **Removing chains:**

❖ **Remove horizontal chains:**

Explanation of approach:

The values of variable chainstart and chainend, which have been recorded in the chain-detecting subroutines, will be passed onto this subroutine (parameters).

Using 2 loops, we can 'shift' the picture codes downwards:

-The pointer k is initially at the first box of the chain (at position *chainstart*)

-The picture code of the box above k will become the picture code box k

PictureArray(k) = PictureArray(k - grid)

- k will then become (k-grid). The pointer k has jumped to the box above. And the process happens again. This loop will end when the pointer has jumped to the top row.

- k is then at the second box of the chain, and the whole process above is repeated. k is then decremented, moving to the next box, and so on. Hence we need an outer loop.

Pseudo- code:

```
For l = chainstart To chainend

    Replace pictures in Box(l) by pictures of smiley faces

Next l

For l = chainstart To chainend

    k = l

   While k > grid

      PictureArray(k) = PictureArray(k - grid)

      k = k - grid

   End while

 Next l
```

k-grid		
k=l		

Picture code of box (k-grid) will become picture code of box k. Then k is decremented. The same process happens.

k-grid		
k		

k is less than grid now, so the loop ends.

k		
	l	

l will be the next box in the chain and k is set to l. When the loop For l= chainstart to chainend has completed:

Start_top_chain		end_top_chain =k

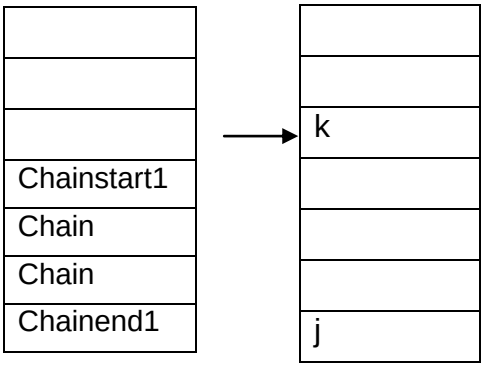
```
   end_top_chain = k
   start_top_chain = end_top_chain - (chainend - chainstart)

         For p = start_top_chain To end_top_chain

            Randomly generate a whole number between 1 and 5 for PictureArray(p)

         Next p

       Call Attach_pictures Subroutine
```

❖ **Remove vertically:**

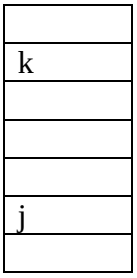
Similarly to removing horizontally, the values of variable chainstart1 and chainend1, which have been recorded in the chain-detecting subroutines, will be passed onto this subroutine (parameters).

Pseudo code:

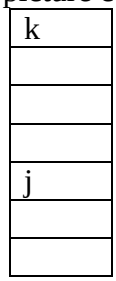
```
For k = chainstart1 To endchain1

   Replace pictures in Box(l) by pictures of smiley faces
   k = k + grid – 1
Next k

 k = chainstart1 – grid

 j = chainend1

   While k > 0
      PictureArray(j) = PictureArray(k)

      j = j - grid
      k = k - grid

   End While
```



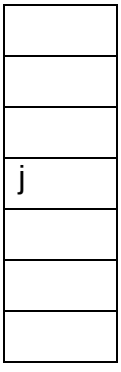
Picture code of box k becomes picture code of box j.



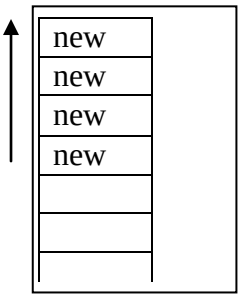
The same process happens- picture code of box k becomes picture code of box j.



Same process happens. j and k are decremented However this time k-grid gives a negative value, hence the loop ends. The diagram on the left shows the final position of j.



```
While j > 0

        Randomly generate a whole number between 1 and 5
        for PictureArray(p)
        j = j – grid
End While

  Call Attach_pictures Subroutine
```



- **Scoring (this will be written within the chain-cheking subroutines)**

Initially the score is 0:
Hence in Form_load, we set the caption of lblScore to 0.

❖ **Changing scores for vertical chains:**

```
If Flag= True Then (flag= true means there has been a chain detected)

  If (chainend1 - chainstart1) = 16 Then

      Set caption of lblScore to (current caption+ 30)
  Else
      Set caption of lblScore to (current caption+ 50)

  End If
```

❖ **Changing scores for horizontal chains:**

```
If Flag= True Then (flag= true means there has been a chain detected)

  If (chainend - chainstart) = 2 Then

      Set caption of lblScore to (current caption+ 30)
  Else
      Set caption of lblScore to (current caption+ 50)

  End If
```

- **Timing:**

```
Set caption of lblCountdown to 120

Set timer's interval to 1000

   If lblCountdown.Caption > 0 Then

      lblCountdown.Caption = lblCountdown.Caption -1


Else

         Disable Timer

         FinalScore = lblScore.Caption

         Show frmTimeUp

         Hide frmPlayGame

End If
```

If lblCountdown.caption is greater than 0 (there is still time left), the caption will be decremented.

(iii) TEST STRATEGY

Test No.	Description	Test data	Expected outcome	Actual outcome
1	**Testing command buttons on Menu Screen**	Button 'Start Game'	Gameplay screen appears. Menu Screen disappears	
		Button 'Quit Game'	A prompt message appears	
		Button 'Instructions'	Instructions screen appears on top of Menu screen.	
2	**Testing command buttons on Instructions screen**	Button 'Start Game'	Gameplay screen appears. Instructions screen (and Menu screen behind) disappears.	
		Button 'Back to Menu'	Instruction screen is closed. Menu screen left behind.	
3	**Testing command button(s) on Gameplay screen**	Button 'Play'	The boxes in the grid are filled in with pictures, one by one. The initial detection and replacement processes start (at this stage **no swap has been made**). Timer starts to run when no more chain is detected.	
4	**Swapping 2 boxes by clicking on 1 box after another**. I will test this function using normal, extreme and invalid test data:			
	Normal	2 horizontally adjacent boxes. Middle of grid	The 2 boxes are swapped	
		2 vertically adjacent boxes. Middle of grid	The 2 boxes are swapped	

	Extreme	2 horizontally adjacent boxes. At right edge of grid	The 2 boxes are swapped	
		2 vertically adjacent boxes. left corner of grid	The 2 boxes are swapped	
	Invalid	Boxes which are 1 box apart	"Bleep" sound	
		Right-most box of a row and left-most box of the next row	"Bleep" sound	
		2 crossly adjacent boxes	"Bleep" sound	
		1 box at the top of grid and the other at the bottom	"Bleep" sound	
5	**Detecting chains of three or more**			
	Normal	3/4/5/6/7 horizontally adjacent boxes with identical pictures inside.	Pictures of a smiley face appear inside the boxes. Counter is set to 2.	
		3/4/5/6/7 vertically adjacent boxes with identical pictures inside	Pictures of a smiley face appear inside the boxes. Counter is set to 2	
	Extreme	8 horizontally adjacent boxes with identical pictures inside	Pictures of a smiley face appear inside the boxes. Counter is set to 2	
		8 vertically adjacent boxes with identical pictures inside	Pictures of a smiley face appear inside the boxes. Counter is set to 2	
		On the same row: 7 boxes with same picture. The 6th and 7th boxes are interrupted by 1 box in between, with different picture.	Pictures of a smiley face appear inside the first 6 boxes. Counter is set to 2	

	Invalid	2 horizontally adjacent boxes with identical pictures.	No effect	
		2 vertically adjacent boxes with identical pictures	No effect	
		3 crossly adjacent boxes with identical pictures	No effect	
		2 right-most boxes of a row and 1 left-most box in next row. Same pictures.	No effect	

6

Replacement of pictures in the boxes when a chain is detected:

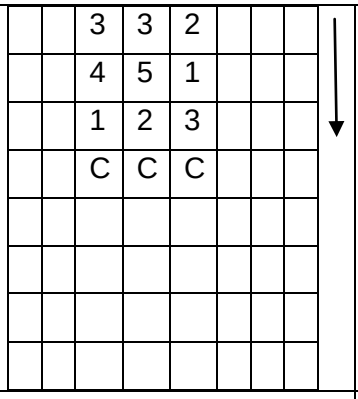
When a chain is detected and, consequently the boxes involved have smiley-face pictures inside, **pictures of the boxes above the chain will 'shift' downwards** (although the visual shifting effect is not included in the game).

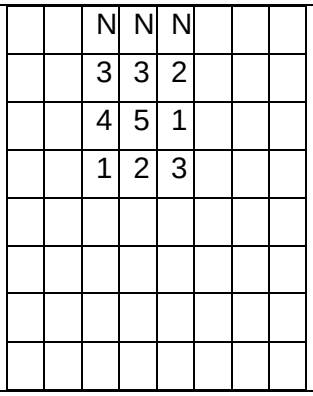
When the shifting has completed, the boxes at the top of the grid will be filled in with new, randomly chosen pictures.

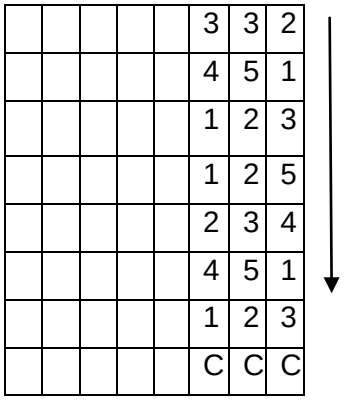
Here I will use diagrams to make the test data and expected outcome clearer. Pictures will be represented by numbers (1 to 5).

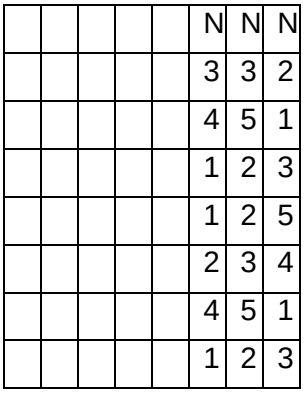
Boxes in the chain will be annotated 'C'.

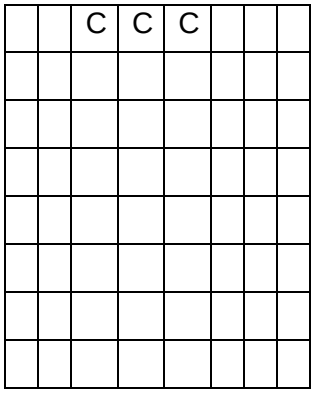
Boxes at the top which will be filled in with new, randomly chosen pictures, are annotated 'N'

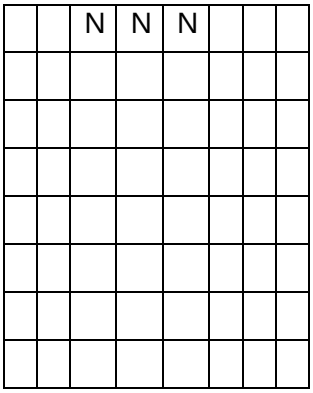


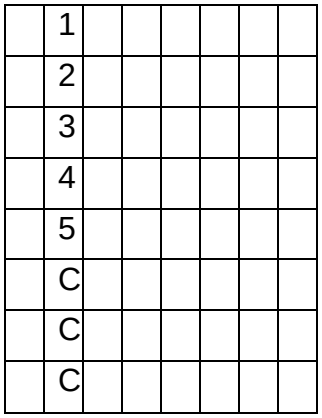


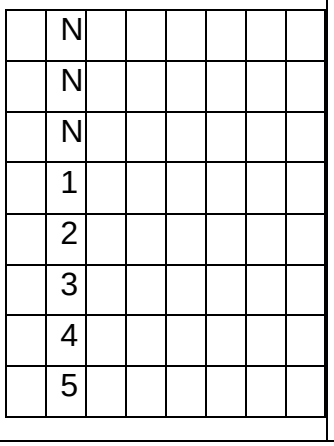


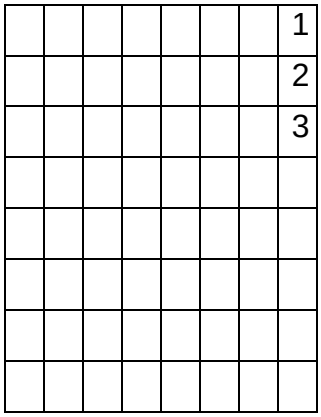
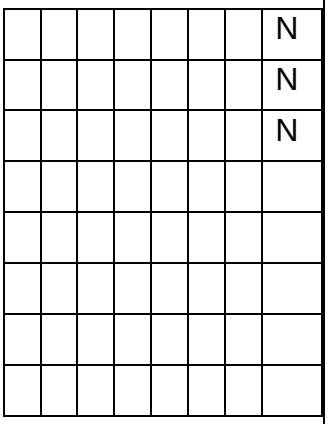










				
7	**Test the Scoring function of the game**	Any chain of 3 created	30 points added	
		Any chain of more than 3 created	50 points added	
8	**Test the timer**	When Gameplay screen appears and button 'Play' hasn't been clicked	Timer stays at 120 (seconds)	
		When button 'Play' is clicked	Timer starts to run backwards. A pop-up message 'Time's Up' appears when timer has come to 0	
9	**Test the final score in Game Over screen**		Final score in Gameplay screen= final score in Game Over screen	
10	**Test command buttons in the Game Over screen**	Button 'Try Again'	Gameplay screen appears.	
		Button 'Quit Game'	Prompt message appears	
		Button 'Back to Menu'	Menu screen appears	
11	**Test command buttons in prompt message box**	Button 'Yes'	All screens are closed	
		Button 'No, Not yet'	The prompt message box is closed	
12	**Test command buttons in pop-up screens**	Message box 'Swaps used up'- button 'OK'	Game Over screen appears	
		Message box 'Time's up'- button 'OK'	Game Over screen appears	
13	**Test the function of counter**	When a swap is made, and no chain is detected	Counter decreased	
		When a swap is made, a chain is detected	Counter is set to 2	
		Counter is 0	'Swaps used up' message box appears	

(c)SOFTWARE DEVELOPMENT AND TESTING

(i) SOFTWARE DEVELOPMENT:

Here I shall provide the final format of each form by using screenshots, together with the fully annotated code that makes the form functional.

Form frmMenu:

This form doesn't have any declarations. Objects include cmdStart, cmdInstruction, cmdQuit, lblMenu, and various images named imgWhale1, imgWhale2, imgFish1, imgFish2, imgStarfish, imgCrab.

Annotated code:

```
Private Sub cmdInstruction_Click()
    frmInstruction.Show
End Sub

Private Sub cmdQuit_Click()
    frmPrompt1.Show
End Sub
```

When cmdInstruction is clicked, this shows the form named frmInstructions

When cmdQuit is clicked, a prompt message box named frmPrompt1 is shown

Private Sub cmdStartGame_Click()

When cmdStartGame is clicked, this shows the form named frmPlayGame and hides the form frmMenu

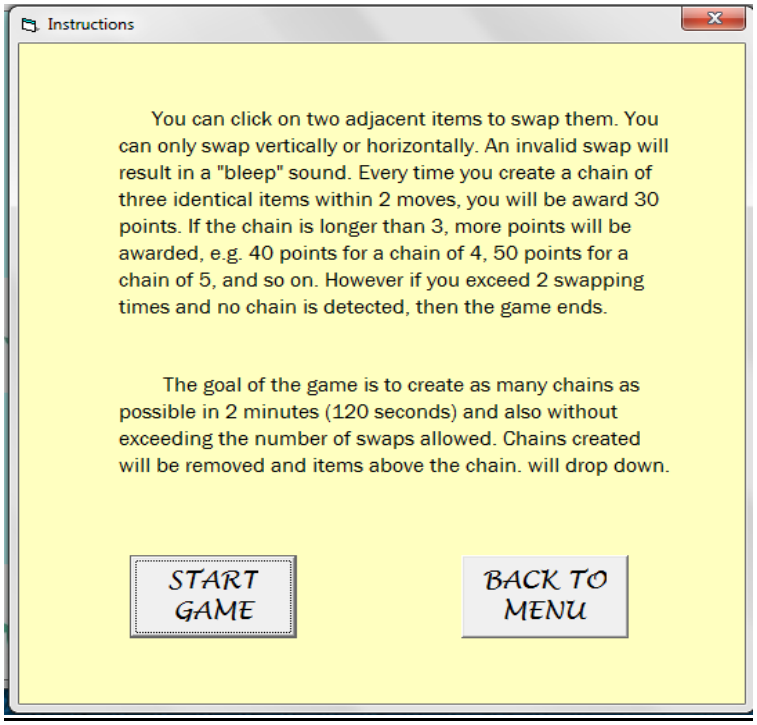
frmPlayGame.Show
frmMenu.Hide

End Sub

Alpha Testing:

I have checked the code by trying all the command buttons on the Menu screen and they all work properly.

Form frmInstructions:



This form does not have any declarations.

Objects contained: lblInstruction1, lblInstruction2, cmdStart, cmdBack

Annotated code:

Private Sub Form_load()

lblInstruction1.Caption= “You can click on two adjacent boxes to swap them. You can only swap vertically or horizontally. An invalid swap will result in a "bleep" sound. Every time there are 3 boxes with identical pictures, 30 points will be awarded. If the chain is longer than 3, more points will be awarded, e.g. 40 points for a chain of 4, 50 points for a chain of 5, and so on. However if you exceed 2 swapping times and no chain is detected, then the game ends.”

lblInstruction2.Caption= “ The goal of the game is to create as many chains as possible in 120 seconds and also without exceeding the number of swaps allowed. Chains created will be removed. Pictures of the boxes above the chain will ‘shift’ downwards (although in the game there is no visual shifting effect).”

End Sub

This fills in the caption of lblInstruction1, lblInstruction2 and lblInstruction3 with instructions for playing the game when this form loads.

```
Private Sub cmdBack_Click()

frmMenu.Show
frmInstruction.Hide

End Sub
```

When cmdBack is clicked, this shows the form frmMenu and hides frmInstruction

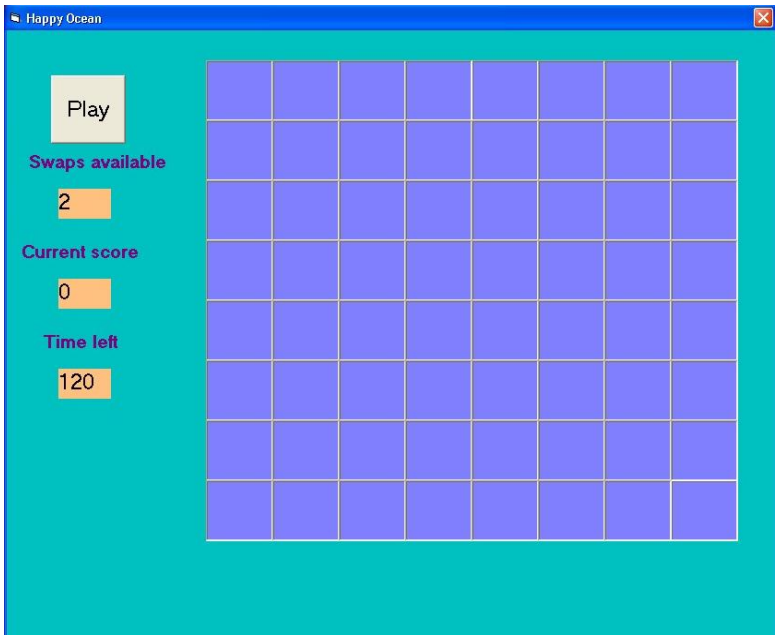
```
Private Sub cmdStart_Click()

frmPlayGame.Show
frmInstruction.Hide
frmMenu.Hide

End Sub
```

When cmdStart is clicked, this shows the form frmPlayGame and hides frmInstruction and frmMenu

Alpha Testing:

I have checked the code by trying all the command buttons on the Instruction screen and the result is that they all work properly.

Form frmPlayGame:



Objects contained: cmdPlay, lblTitle1, lblTitle2, lblTitle3, lblCounter, lblScore, lblTimeleft, and a control array of picture boxes named Box.

Declarations:

```
Dim PictureArray(1 To 64) As Byte
'stores the picture codes of 64 boxes in the grid

Dim flag As Boolean                    'used to flag up if a box is clicked on
Dim flag1, flag2 As Boolean
 'used to flag up if respectively, a horizontal is detected and a vertical chain is detected

Dim click1, click2 As Byte          'stores the indices of the 2 pictures being clicked
Dim grid, score As Byte             'stores the value for size of grid and current score
Dim j, k, l, p, n, m As Byte        'general constants, reused in many subroutines

Dim hor_start, hor_end As Byte
'stores the indices of the starting box and the ending box of a horizontal chain

Dim ver_start, ver_end As Integer
'stores the indices of the starting box and the ending box of a vertical chain

Dim possiblemoves(1 To 4) As Integer
```

```
‘stores the set of box indices that will make a swap valid (one box has already been clicked, this is for the second box)

Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
‘pause the program in milliseconds)

Public FinalScore As Integer
‘stores the final score. This variable can be used by another form (frmGameOver)
```

Subroutine Form_load and its command button:

Private Sub Form_load()

```
    flag = False          'no picture has been clicked on
    flag1 = False         ‘no horizontal chain has been detected
    flag2 = False         ‘no vertical chain has been detected
    Timer.Enabled = False
‘This sets the timer to be inactive- the timer only starts when button cmdPlay is clicked

    grid = 8       ‘This sets the size of grid to be 8

    Score = 0    ‘Score is set to 0

    Randomize

    For p = 1 To 64

        PictureArray(p) = Int(Rnd * 5) + 1

    Next p
```

All the Booleans are set to False to make sure none is holding a value from a previous run of the program

This randomly creates a picture code for each box in the grid. The numbers generated from the calculation Int(Rnd*5)+1 are integers between 1 and 5.

End Sub

Private Sub cmdPlay_Click()

```
        Timer2.Enabled = True

        Call Attach_pictures
```

When cmdPlay is clicked, this activates the timer- time starts to run from 120 (seconds).This also calls the subroutine Attach_pictures.

End Sub

Alpha Testing:

I have clicked on the button Play and the result was that the grid is filled in, box by box, with random pictures numbered 1 to 5.

Subroutine Attach_pictures

This subroutine attaches pictures to the boxes at the beginning of the game when cmdPlay is clicked and update pictures whenever subroutine Play_Click is called. This process of putting pictures onto the boxes is done by having a loop around the grid (from box 1 to box 64).

When picture code is changed in **some boxes**, the subroutine is called and **every box** would be replaced by a new picture (even though this new picture can be the same as the previous). **So this condition below ensures that only when the new picture is different to the current picture in the box then the attaching process happens.**

Private Sub Attach_pictures()

```
For p = 1 To 64

    Select Case PictureArray(p)

      Case 1

        If Box(p).Picture <> PicAnimal1.Picture Then
            Box(p).Picture = PicAnimal1.Picture



            'If picture code is 1, show picture PicAnimal1

            Sleep (100)

        End If

      Case 2

        If Box(p).Picture <> PicAnimal2.Picture Then
            Box(p).Picture = PicAnimal2.Picture
            ' if picture code is 2, show picture PicAnimal2
            Sleep (100)
        End If

      Case 3
        If Box(p).Picture <> PicAnimal3.Picture Then
            Box(p).Picture = PicAnimal3.Picture
            'if picture code is ,3 show picture PicAnimal3
            Sleep (100)
        End if

      Case 4
```

This line of code delays the program by 100 milliseconds. It helps slow down the program so the user can see more clearly the attaching process.

```
            If Box(p).Picture <>PicAnimal4.Picture Then
            Box(p).Picture = PicAnimal4.Picture
            'If picture code is 4, show picture PicAnimal4

            Sleep (100)
            End If

        Case 5

            If Box(p).Picture <> PicAnimal5.Picture Then
                Box(p).Picture = PicAnimal5.Picture
                'If picture code is 5, show picture PicAnimal5

                Sleep (100)
            End if

    End Select

Next p

Call Checking_horizontal
```

Call subroutine named Checking_horizontal

End Sub

Alpha Testing:

At first the code above did not have the condition

If Box(p).Picture<>Pic1.Picture then, If Box(p).Picture<>Pic2.Picture, etc.

After running the code I realised that even when the new picture was the same as the current picture, the attaching process still happened and there was a visual effect of it (like a blink). Moreover the program may be slowed down by changing the pictures of every single box in the grid. So the code was added with the condition and the final code has been shown above.

Subroutine Box_Click:

When any box is clicked, this subroutine records the box's index, flags up if the box is firstly-clicked, and works out other boxes that can be clicked next to make the swap valid. When the second box is clicked, this subroutine checks the box's index against the working values that it has, decides whether a valid swap has been made and finally does the swapping process.

Index is the parameter in this subroutine. The grid is made of a control array of picture boxes named Box, and each of them has an index. When a box is clicked on, the system will record the index of the box. This value can be passed into the program.

```
Private Sub Box_Click (Index As Integer)

Dim Temp As Byte

If flag = False Then
```

The value *Index* will change when another box is clicked, so we have to store the intial value in click1

```
    click1 = Index

    remainder1 = click1 Mod grid

    Select Case remainder1
```

This calculation decides whether the box is at the left/right edge of the grid. If remainder1 =0, the box is at the left edge. If remainder1=1, the box is at the left edge. If remainder1 is neither 1 nor 0, this is a normal case- the box is at the middle of grid. This step is important because it determines the indices of boxes that can be clicked next which would make a valid swap.

```
        Case 1

        possiblemoves(1) = click1 + 1
        possiblemoves(2) = click1 + grid
        possiblemoves(3) = click1 - grid
        possiblemoves(4) = 0
```

Remainder1=1, box is at left edge. Hence the 3 boxes that can be clicked next which would make a valid swap are: above, to the right and below the firstly-clicked box. The array possiblemoves consists of 4 items so we need to assign the 4th item to 0

```
    Case 0

        possiblemoves(1) = click1 - 1
        possiblemoves(2) = click1 + grid
        possiblemoves(3) = click1 - grid
```

Remainder1=0, box is at right edge. Hence the 3 boxes that can be clicked next which would make a valid swap are: above, to the left and below the firstly-clicked box. Similarly, the 4th item of the array needs to be assigned to 0

```
        possiblemoves(4) = 0

    Case Else

        possiblemoves(1) = click1 + 1
        possiblemoves(2) = click1 - 1
        possiblemoves(3) = click1 + grid
        possiblemoves(4) = click1 - grid


  End Select

  flag = True ' This sets flag to True- there has been a click
```

Remainder1 is not 0 or 1, box is at the middle of grid. Hence the 4 boxes that can be clicked next which would make a valid swap are: above, below, to the right, to the left of the 1st box.

'The following lines of code swap the two boxes that have been clicked, first by swapping the picture codes in the array, then by updating the pictures in the boxes to match the array:

```
Else    'If flag= True

  click2 = Index
```

The index value will change when another box is clicked, so we store the index value of secondly-clicked box (or box 2) in click2

```
  For p = 1 To 4

    If possiblemoves(p) = click2 Then

      Temp = PictureArray(click2)

      PictureArray(click2) = PictureArray(click1)

      PictureArray(click1) = Temp

    p = 4

    lblCounter.Caption = lblCounter.Caption - 1   'Counter is decremented
```

If the index of box 2 equals to any one of the indices stored in the array possiblemoves, it means that the swap is valid. The system will store the picture code of box 2 in Temp.

Picture code of box 2 is set to picture code of box 1. Now these two boxes have the same picture code.

Picture code of box 1 is set to picture code of box 2, which is stored in Temp.

```
        Call Attach_pictures

        If lblCounter.Caption = 0 Then
          frmUseUp.Show
          frmPlayGame.Hide
          FinalScore = lblScore.Caption
        End If
    End If
  Next p

  flag = False 'Flag is set back to False
```

This calls the subroutine Attach_pictures so that pictures are updated to match the array. The process of checking for chains of 3 (or more) will be called within Attach_pictures.

If counter becomes 0, the game is over. So this shows message box frmUseUp, hides frmPlayGame and sets public variable FinalScore to the current score in frmGamePlay

End If

End sub

Alpha testing:

When I first completed this code and ran it, there was a slight problem of getting the current score in form frmPlayGame to become the final score in form frmGameOver, since variables in one form can't be used in another form. I have fixed this problem using a public variable FinalScore- a public variable can be used in various forms. So the final version of the code has been shown above.

Chain-Checking subroutines

These subroutines are used to detect horizontal and vertical chains of three or more. The chain-removing and replacing process will be called within these subroutines.

I use 2 pointers to keep track of the starting point and ending point of the chain**:** *m* acts as a starting pointer while n acts as the ending pointer. n will be incremented each time if the picture codes of box m and n are the same.

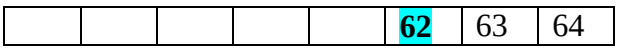
Condition l=0 is made up to ensure that when pointer n reaches the right edge, the search has to stop- incrementing n further would lead to a box in the next row and it cannot be counted as part of the chain.

The following subroutine Checking_horizontal is used to detect horizontal chain:

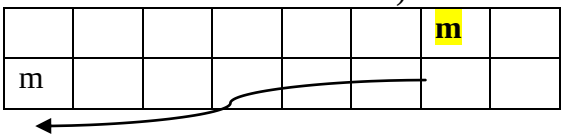
Private Sub Checking_horizontal()

```
chainstart = 0          'Initialise the variables

endchain = 0


l = 0

For m = 1 To ((grid * grid) - 2)

   For k = 1 To (grid - 1)
      If m = (grid * k) - 1 Then m = (grid * k) + 1
   Next k


   n = m + 1 'The ending pointer is incremented

  While l = 0 And PictureArray(m) = PictureArray(n)

      If n Mod grid = 0 Then

          l = 1   'to exit the loop
      Else
          n = n + 1
      End If

  Wend
```

In this diagram, the numbers represent the indices of the boxes.
We just need to loop from m=1 to 62, because there is no more chain that can be made when n>62

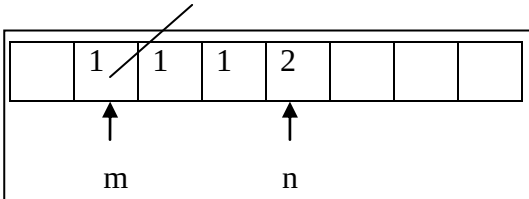


If the starting pointer m is 1 box away from the edge then m is set to the first box of next row (there is no need to search further, since a chain must have at least 3 boxes)



This ensures that when the ending pointer n has reached the edge then the search has to stop. If it hasn't, we increment n when the picture codes of m and n are the same

Picture code is different→ stop the search



So in order to get the right starting point and ending point of the chain, we have to decrement n. However in the case of n reaching the edge of grid (l=1), n is the ending point of the chain.

```
If l = 0 Then

   n = n – 1
End If
```

By taking (n-m), this determines whether a chain has been made. If (n-m)>=2, this means that the chain is 3 or over in length, making it valid. Hence this sets Flag1 to True and stores the values of m and n in variable chainstart and chainend (because m and n will be altered when the program continues to run)

```
 If (n-m) >= 2 Then

     Flag1 = True

     chainstart = n

     chainend = m

     m = (grid * grid) - 2    'to end the outer loop

 End If

Next n

   If Flag1 = True Then      ' if there is a chain detected

     If (chainend-chainstart) = 2 Then

        lblScore.Caption = lblScore.Caption + 30

     Else
        lblScore.Caption = lblScore.Caption + 50
     End If

     lblCounter.Caption = 2

      Call Remove_horizontal(chainstart, endchain)
     'This calls subroutine Remove_horizontal
```

This deals with the scoring system. If the chain involves 3 boxes (chainend-chaintstart=2), caption of label lblScore will be incremented by 30. If the chain is longer than 3, caption of label lblScore will be incremented by 50.
Counter will also be set back to 2

```
    Else
        Call Checking_vertical
'If there is no horizontal chain detected, subroutine Checking_vertical is called

    End If

End Sub
```

Alpha Testing:

I used command Print in various places in the code and ran the program. Therefore I could narrow the area where some problems occurred, which made debugging a lot easier.

e.g. 1)

```
Private Sub Checking_horizontal()

chainstart = 0
endchain = 0

Print "checking_horizontal"
```

This was to check whether this subroutine has been executed.

2)

```
If Flag1 = True Then
   Print "chainstart=", chainstart
   Print "chainend=", chainend
…
```

This was to test the values of chainstart and chainend- they were consistent with the values I worked out from the screen and also with the values worked out from previous lines of code.

The following subroutine Checking_vertical detects vertical chains and calls subroutine Remove_vertical at the end. In the diagrams below m is the starting pointer and n is the ending pointer.

Private Sub Checking_vertical()

```
flag2 = False
n = 0
m = 0
l = 0
```

Initialise the variables

For m = 1 To 48

```
    n = m + grid

    While l = 0 And PictureArray(n) = PictureArray(m)

        If n < 57 Then

            n = n + grid
        Else

            l = 1
        End If

    Wend
```

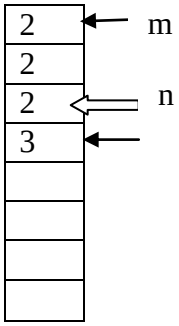
	m		m				
49			n				56
57	n	59	60	61	62	63	64

When the 2 conditions l=0 and same picture code are met, if n<57 we increment n (n= n+ grid).

When n>57 (an example is shown in the diagram), if we increment n we will get a value out of range (66>64). So in this case we do not increment n. l is also set 1 to end the loop since no more box can be added to the current chain.

```
If l = 0 Then
    m = m - grid

End If
```

Similarly to Checking_ horizontal, this step finds the correct ending point of the chain. The numbers in the diagram below represent the picture codes of different boxes.



Picture code is different, so stop the search. In order to get the correct ending point of the chain, we have to 'backtrack'- decrease n by 8.

However in the case when m>57 we do not increment n and set l=1, so at this stage e need not to decrement m.

```
        If (n - m) >= 16 Then

            flag2 = True

            chainstart1 = m

            chainend1 = n

                m = 48   'to end the loop
        End If
Next n


   If flag2 = True Then
      If (chainend1 - chainstart1) = 16 Then

            lblScore.Caption = lblScore.Caption + 30
      Else
            lblScore.Caption = lblScore.Caption + 50
     End If

        lblCounter.Caption = 2

'A chain has been made so the player is left with 2 available swaps. Caption of label
lblCounter is set back to 2

        Call Remove_vertical(chainstart1, chainend1)
  'This calls subroutine Remove_vertical with chainstart1 and chainend1 as parameters

   End If

End Sub
```

By taking (n-m), this determines whether a chain has been made- If (n-m)>=16, it means that the chain is 3 or over in length, making it valid. So we set Flag2 to True and store the values of m and n in variable chainstart1 and chainend1.

If the chain involves 3 boxes (chainend-chaintstart=16), caption of label lblScore will be incremented by 30.

If the chain is longer than 3, caption of label lblScore will be incremented by 50. Counter will also be set back to 2

Alpha Testing:

Similarly to the Checking_horizontal subroutine, I put in various Print commands in the code to test. In addition I recorded the scores appearing on screen before and after the creation of a chain and therefore knew that the code worked.

Chain-removing subroutines

In these subroutines the chain detected in Chain-Checking subroutines will be removed. Pictures of boxes above the chain will fall down (there is no visual effect of falling items). Then, the top row boxes will need to be filled in with new pictures, which can be done by creating random picture codes and then calling the subroutine Play_Click to update the pictures.

Private Sub Remove_horizontal (ByVal chainstart As Integer, ByVal endchain As Integer)

Chainstart and chainend are parameters; the values of them are passed from the subroutine Checking_horizontal to this subroutine

```
Flag1 = False          'Sets Flag1 back to False

For l = chainstart To chainend

    Picture1(l).Picture = smileyface.Picture

Next l

Sleep (200)
'Pause the program for 200 milliseconds


For l = chainstart To chainend

    k = l

   While k > grid

      PictureArray(k) = PictureArray(k - grid)

      k = k - grid

   Wend


Next l
```

This replaces all the pictures of the boxes in the chain with pictures of a smiley face. This is to create a visual effect to the player.

k-grid		
k=l		

Picture code of box (k-grid) will become picture code of box k. Then k is decremented. The same process happens.

k-grid		
k		

k is less than grid now, so the loop ends.

k		
	l	

l will be the next box in the chain and k is set to l. When the loop For l= chainstart to chainend has completed:

Start_top_chain		end_top_chain =k

```
    end_top_chain = k

'the final value of k (after the loops) is
the end of the top chain that needs to be
filled in with new pictures

    start_top_chain = end_top_chain - (chainend - chainstart)

    'Start_top_chain is found using end_top_chain taking away (chainend-chainstart)


        For p = start_top_chain To end_top_chain

            PictureArray(p) = Int(Rnd * 5) + 1

        'this generates random picture codesbetween 1 and 5

        Next p

        Call Attach_pictures    'This calls subroutine Attach_pictures
```

End Sub

Alpha Testing:

1) I monitored variables by setting up lots of Print commands within the subroutine e.g.

Private Sub Remove_vertical(ByVal chainstart1 As Integer, ByVal endchain1 As Integer)

'Print "remove_vertical"

'Print "endchain1", endchain1
'Print "chainstart1", chainstart1

This was to make sure the parameters were passed correctly from the subroutine Checking_vertical

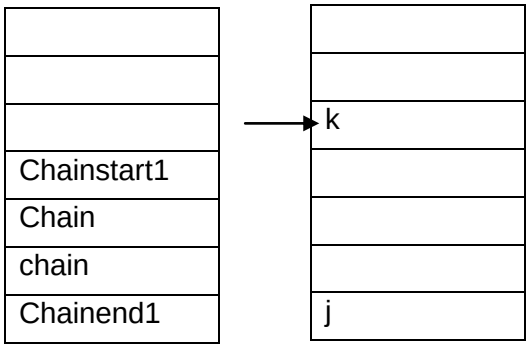
2) I tested many cases of the game by running it lots of times and recording the final pictures inside the boxes. The code was functional.

Private Sub Remove_vertical (ByVal chainstart1 As Integer, ByVal endchain1 As Integer)

```
For k = chainstart1 To endchain1

    Picture1(k).Picture = smileyface.Picture

    k = k + grid - 1

Next k

Sleep (200)

'Pause the program for 200 miliiseconds
```

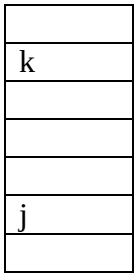
This replaces all the pictures of the boxes in the chain with pictures of a smiley face (visual effect for the player)

```
k = chainstart1 – grid

j = chainend1
```

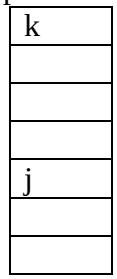


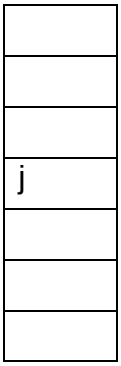

```
While k > 0

    PictureArray(j) = PictureArray(k)

    j = j - grid
    k = k - grid

Wend
```

Picture code of box k becomes picture code of box j.



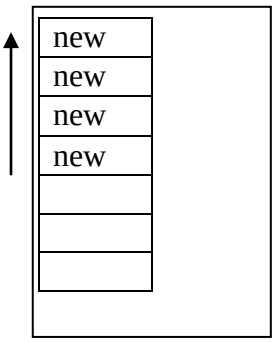
The same process happens- picture code of box k becomes picture code of box j.





Same process happens. j and k are decremented However this time k-grid gives a negative value, hence the loop ends. The diagram on the left shows the final position of j.

```
While j > 0

    PictureArray(j) = Int(Rnd * 5) + 1
    j = j – grid

    'This generates random picture codes from 1 to 5

Wend

  Call Attach_pictures      'This calls subroutine Attach_pictures
```



End Sub

Alpha Testing:

I ran a small part of the code each time to identify the problems

1) I ran the first part of the code and realized the positions of the smiley faces were not correct- they did not replace the right vertical chain. This was the original piece of code:

```
For k = chainstart1 To endchain1

   Picture1(k).Picture = smileyface.Picture

   k = k + grid

Next k
```

The reason was that the For…next loop would increment k after each round, so the second and third boxes of the vertical chain would be shifted 1 box to the right, making the pictures of smiley face being in the wrong position. I then made changes to the code to shift these boxes back to the right place:

```
For k = chainstart1 To endchain1

   Picture1(k).Picture = smileyface.Picture

   k = k + grid - 1

Next k
```

2) I tested many cases of the game by running it lots of times and recording the final pictures inside the boxes. The code was functional.

Timer subroutine:

The timer's interval has been set to 1000 (in milliseconds), which means every 1 second this subroutine will be called. When the subroutine is called, the number filled in the caption of label lblCountdown will be decremented. This creates the effect of a countdown timer.

```
Private Sub Timer2_Timer()

    If lblcountdown.Caption > 0 Then

        lblcountdown.Caption = lblcountdown.Caption -1

Else

          Timer2.Enabled = False

           FinalScore = lblScore.Caption


          frmTimeUp.Show

          frmPlayGame.Hide

End If

End Sub
```

If lblCountdown.caption is 0 (time has run out), the game is over. So Timer2 is disabled, current score (which is the caption of lblScore) will be stored in public variable FinalScore. Form frmTimeUp will be shown and form frmPlayGame will be hidden.

Alpha Testing:

Once I pressed the Play button and the initial chain-checking and chain-removing finished, the number in the timer box started to get smaller each second. When it came to 0, a form popped up saying "Time's Up" (although the number 0 couldn't be seen on screen). So the subroutine has satisfied the user requirements.

Form frmUseUp:

This form informs the user that he/she has used up 2 swaps without making any chain. It does not have any declarations and contains label named lblUseUp, command button cmdOK.

Annotated code:

Private Sub Form_load()

lblUseUp.caption = “Sorry, you have used up 2 swaps without creating any chain!”

End sub

Private Sub cmdOK_Click()

frmGameOver.Show
frmUseUp.Hide

‘This shows form frmGameOver and hides form frmUseUp

End Sub

Form frmTimeUp:

This form informs the user that time has run out. It doesn't have any declarations and include label named lblTimeUp and command button cmdOK

Annotated code:

Private Sub Form_Load()

lblTimeUp.Caption= 'TIME'S UP'

End sub

The code above fills in the caption of lblTimeUp with the message for the player when time has run out.

Private Sub cmdOK_Click()

frmGameOver.Show

frmTimeUp.Hide

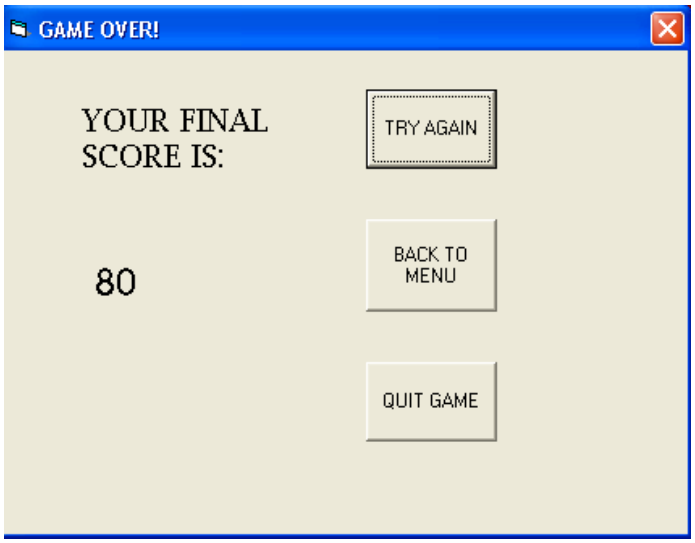
'This shows form frmGameOver and hides form frmTimeUp

End Sub

Form frmGameOver:

This form does these tasks: show the final score to the user and list 3 options to the user: Try Again, Back to Menu and Quit Game.

This subroutine does not have any declarations. Objects include lblGameOver, lblFinalScore, cmdTryAgain, cmdBack, cmdQuit



Annotated code:

Private Sub Form_Load()

lblGameOver.Caption= “Your final score is:”

‘Fill in the caption of label lblGameOver

lblFinalScore.Caption = frmPlayGame.FinalScore

FinalScore is a public variable that can be used in different forms. Here, the caption of label lblFinalScore is filled in with the variable FinalScore taken from form frmPlayGame. This is the score when the game ends.

End sub

Private Sub lblBackMenu_Click()

frmMenu.Show
frmGameOver.Hide

This shows the form frmMenu and hides form frmGameOver

End Sub

Private Sub TryAgainButton_Click()

Unload frmPlayGame
Load frmPlayGame

frmPlayGame.Show
frmGameOver.Hide
End Sub

Previously, form frmPlayGame was only hidden but it still contains the previous run of the game. When button cmdTryAgain is clicked frmPlayGame has to be unloaded and reloaded again to present a new game for the user. Then it will be shown to the user. frmGameOver will be hidden.

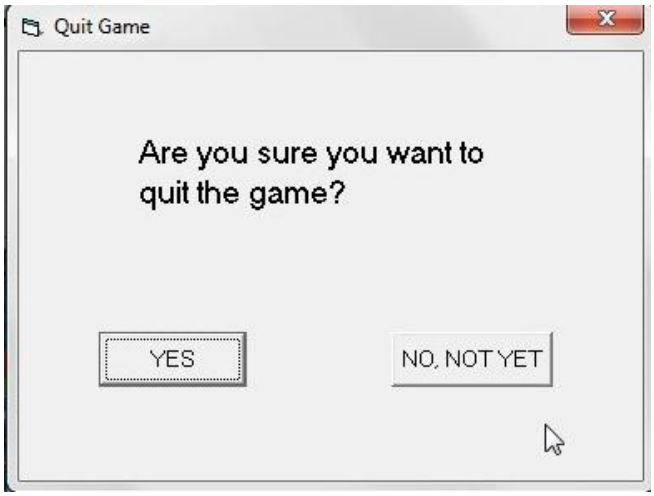
Private Sub cmdQuit_Click()

frmPrompt2.Show

End Sub

This shows the prompt message "Are you sure you want to quit"?

Form frmPrompt:



When this form is loaded, it will ask the user whether he/she wants to quit the game. If "Yes" is clicked, all forms will be unloaded. If "No, not yet" is clicked, the prompt message box will disappear, leaving the other form behind (frmMenu or frmGameOver)

This form does not have any declarations. Objects include lblPrompt, cmdYes, cmdNo

Annotated code:

Private Sub No_Click()

frmPrompt.Hide

'Hides form frmPrompt

End Sub

Private Sub Yes_Click()

Unload Me
End
End Sub

Unloads all forms and ends the program

Alpha testing for form frmUseUp, frmTimeUp, frmGameover and frmPrompt:

I have tried all the buttons on the forms and they all work.