ACKNOWLEDGEMENTS

It has been a great pleasure to work with different people, to share the ideas and existing the knowledge from their ideas throughput he project period. We like to express our gratitude to our **HOD** (Head of Department) who appreciated us to work on this project. We would like to thank to Sir **Babu Ram Dawadi,** who supervise the project and appreciated us to work on this project. Similarly, we would also like to thank **Suraj Singh, Keshab Raj Joshi** and **Kamal Nepal** sir for teaching his special way for doing any kind of tasks and since then his words have always proven to be golden .His way of making us do all the rough work and combine them as the essentials of the projects was a very good concept indeed. We personally am now aware how benefitted we am with this incredible method of learning.

In addition to all the help from my teachers, we received a lot of help and support from my friends. Their suggestions and comments have been most helpful for the development of the program and they deserve a significant part of the credit that goes to this program.

Last but not the least; we will always be indebted to Sir **Babu Ram Dawadi** for inspiring us to develop programs in the very first place.

INTRODUCTION

This project is a mini concept to the **Tetris game (Falling Blocks)** .With a graphics this is in an interesting game to play and we have focused on the logic as well as the outlook. This game has features like every other game available in our nearby's. Also we have included in this project the extended concept of modular programming i.e. we have divided the project into number of functions so that whoever wants to study the logic may prove this useful. Similarly, this project has also the feature of data file where anyone can save their name after they had played game and eventually the file will have stored the player's name, score, cleared level, speed, time and date. The user can also view the past records of the players who played the game .Thus all in all we have tried to make this project like a real one and user friendly too.

Background

This project was primarily designed for a semester project for "Computer Programming ". This project, as its title "**Tetris (Falling Blocks)"** suggests, is an attempt to develop a game that is not only a game but it's an approach to create much more complicated software like these.

I will not boast that the project is complete by itself and bug-free at its present version; however, I believe it can serve as a starting point for future who want to develop more interactive and user friendly software.

In our point of view we find students playing various kinds of games built with a very good graphics and control. Well at the same time they are cheering about the games they forget their potentiality that they are also able to build such kind of games. Well we are just trying to show them that we can also develop such kind of software and gaming stuffs but with right kind of dedication and logical thinking.

I sincerely hope the program fulfils its purpose and proves helpful to anyone who might use it on the entertainment basis as well as logical basis.

Anyone wishing to improve and upgrade the program is very welcome to do so.

Developers: - **Keshab Bahadur Sunari** and **Kiran Bohaju**

Problem Analysis

Simply, Problem Analysis means identifying and studying the problems regarding the program in detail. Moreover Problem Analysis is the foundation for a good program and is therefore an important part of program development cycle.

It is defined as the stage at which the programmer begins roughing out the ideas and logic that he/she will use when the actual coding begins. Thus problem analysis to be used in writing the program, in order to achieve the solution of a problem.

As a part of academic evaluation, we have to develop a software on C programming language on one selective title. Then, we decided to develop a game named **"Tetris"** i.e. falling blocks game on our C project. Here we have identity following problems which may encounter during coding.

- **Making Bricks/Blocks of different shape and color**: This is the beginning stage of the program we have to show a falling brick on the screen in this game. For making bricks, we have to use different colors form c graphics header file library functions. Also the bricks should look like 3-dimensional in appearance. In addition, bricks of different shapes and color should be design in which we decided to use the concept of matrix for different type's blocks. We will assign shape of bricks as following: **I, L, L2, S, S2, T** and **O.**
- **Displaying screen boundary for falling blocks:** Likewise another problem is to show the main screen boundary in which bricks starts to fall. We can either use the graphics header file library function rectangle or draws four different lines for displaying boundary.
- **Checking Detect Collision:** Similarly, we have to make sure that the falling bricks mores left or right within the boundary. It means, when player moves brick to left or right side, it should not go out of left or right boundary line respectively. Moreover, brick should stop moving down after the collision with either lower boundary line or other bricks.
- **Random Shape and Random Color:** When the game is running, among the seven different shape we have to get different bricks with different color one after next time.
- **Next brick and rotation:** We have to design another smaller screen area in which we have to show the brick which is coming up next. Meanwhile, when bricks are falling, players may need to rotate the bricks according to their wish in order to fill space among falled bricks.
- **Removing bricks (To cleared one row of blocks) and counting removed line:** As all the small bricks come and fill the horizontal lines in the screen area, we have to remove such lines in blinking sense looked like a destroy of one or more than one complete horizontal bricks (suppose one complete horizontal bricks is equal to combination of 15 bricks). At the same time, we have to count the number of lines of block cleared.

- **Score and Speed level Analysis:** Another problem is we have to increase the score as the speed of falling blocks increases downwards. We need to increase score in proportion to the rate of pressing of downward arrow key. In addition, when one horizontal line is formed with the combination of brick, the line is removed and score are counted and level be increased. Example: when bricks are removed, at the same time score is increased by 100 points. As the score crossed every 2700 points, speed increase by 5 and level also increase by 1.
- **Special keys:** In this game, we need to design a coding in such a way that a player may paused the game at any time and play by simply pressing any key. Similarly, player must be facilitated with escape option if he wants to exit. In addition, arrow heads should be used a direction moving keys for game also we used 'p' or 'P', 'Esc', 'Shift' and 'Enter' keys. We will used for those key as follow: arrow up head, space and enter key for to rotate blocks, left arrow head for to left side, right arrow head for to right side, down arrow head for to speed up the falling blocks, Esc for to exit from game and p or P for to pause the game.
- **Getting Player's name:** After running the program, a message box should be displayed in which player could enter his/her good name.
- **Record:** We have to make a record file in which name, score, speed and level of each player is stored and can be accessed any time when we needed. If player wants to see past records, we should shows them the records of previous players as will.

Development

In this project, we try to implement a **Tetris Game** using programming language **Turbo C++**. This program comes with various turbo C graphics function. So we try to explain all the tricky parts of code inside it, for those who are not familiar with the common functions and other simple contents of turbo C. The brief description given below explains all the tricky parts which are used in this project.

1. **The graphics.h file of Header (Header file):**
   All the Turbo C++ graphics functions require the graphics.h header file to be included in the program. This file contains prototypes for these functions. Because many graphics functions use constants instead of numbers to specify such values as colors, patterns, and fonts, the graphics.h file also contains definitions for these constants. We won't get far in Turbo C++ graphics programming if we don't include this file.
   i.e. #include<graphics.h>

2. **Initgraph:**
   Initgraph function is used to initialize with the graphics library and changes to the graphics screen for drawing. It is the first step we need to do during graphics programming.

3. **Closegraph:**
   Closegraph function is used to reset back to text mode screen or in other words it used to exit from the graphics screen.

4. **Setpalette:**
   Setpalette function is used to make changes to only one color. It will accept the current color and the destination color.

5. **Getimage:**
   Getimage is the function used to copy a portion (rectangle) into the memory. It accepts 5 parameters – left, top, right, bottom and image pointer (void far*). Sufficient memory needs to allocate.

6. **Putimage:**
   Putimage function is to specify the point (x, y) where needs to be drawn and pointer of the image buffer and operator- type of copy.

7. **Imagesize:**
   Imagesize is the function used to calculate the image size of the given rectangle. Once the image is captured using getimage, we can use putimage to plot the image.

8. **Fillpoly:**
   Fillpoly function is used to draw the polygon with any color filled and/or style set by setfillstyle function.

9. Setfillstyle:

Setfillstyle function is takes input arguments – style and color.

10. Cleardevice:

Cleardevice is the function used to clear the whole screen with the background color.

11. Rectangle:

Rectangle is the graphics function used to draw a rectangle by accepting left, top, right and bottom position. It draws the rectangle in the color set by setcolor function.

12. Setcolor:

Setcolor function is used to set the foreground color in graphics mode.

13. Outtextxy:

Outtextxy function which takes x-position, y-position and string for displaying at a particular location.

14. Delay:

The delay ( ) function found in dos.h is processor dependent. And it won't work on all systems. The reason is the delay function is implemented with clock speed. Delay function takes a single parameter: the time in milliseconds. For a delay of one-half, for example, we would use 500. The value 20 used in the program is so short it produces only a click.

15. Kbhit:

Kbhit function is used to determine if a key has been pressed or not. To use kbhit function in your program you should include the header file “conio.h”. If a key has been pressed then it returns a non-zero value otherwise returns zero.

16. Gotoxy:

Gotoxy function places cursor at a desired location on screen i.e. we can change cursor position using gotoxy function.

17. Sound:

Sound function produces the sound of a specified frequency. Used for adding music to c program, try to use some random values in loop, vary delay and enjoy.

18. Nosound:

Nosound function turn off the PC speaker.

19. Getch:

Getch function prompts the user to press a character and that character is not printed on screen, getch header file is conio.h

20. Line:

Line function is used to draw a line in the graphics screen. It uses the color and style set by the setcolor and setlinestyle functions.

21. Clearviewport:

Clearviewport erases the viewport and moves the current position to home (0, 0), relative to the viewport.

22. Randomize:

Randomize initializes the random number generator with a random value.

23. Restorecrtmode:

Restorecrtmode restores the original video mode detected by initgraph. This function can be used in conjunction with setgraphmode to switch back and forth between text and graphics modes.

24. Bar:

Bar function is to draws a filled-in, rectangular, two-dimensional bar. The bar is filled using the current fill pattern and fill color. Bar does not outline the bar.

25. Malloc:

Malloc allocates a block of size bytes from the memory heap. It allows a program to allocate memory explicitly as it's needed, and in the exact amounts needed.

26. Gets:

Gets collects a string of characters terminated by a new line from the standard input stream Studio In Out and puts it into character type's variable or name.

27. Setlinestyle:

Setlinestyle function is used to set the linestyle and it accepts 3 parameters as input. The first one is the line style it can be SOLID_LINE, DOTTED_LINE, CENTER_LINE, DASHED_LINE or USERBIT_LINE. The second argument is to use the user bit pattern and the last one is line thickness it can be 1 pixel or 3 pixel wide.

28. Rand:

Rand function is uses a multiplicative congruential random number generator with period 232 to return successive pseudo-random numbers in the range 0 to maximum value returned by rand function.

29. Settextstyle:

Settextstyle sets the text font, the direction in which text is displayed, and the size of the characters.

30. Puts:

Puts copies the null-terminated string character type's variable or name to the standard output stream Studio Out and appends a newline character.

31. Getc:

Getc function is used for to reads a character from a file.

32. EOF:

EOF determines whether the file associated with handle has reached end-of-file.

33. FILE:

File control structure for streams.

34. Fopen :

Fopen function is used for to create a new file for use. Opens and existing file for use.

35. Fprintf:

Fprint function is used for to write a set of data values to a file.

36. Fclose:

Fclose function is used for to close a file which has been opened for use.

37. Time_t:

This variable type defines the value used for time function.

38. Global variable:

The variables that are defined outside any function are called global variables. All functions in the program can access and modify global variables. It is useful to declare a variable global if it is to be used by many functions in the program. The default initial value for this variable is zero

39. Macros:

A macro is a fragment of code which has been given a name. Whenever the name is used, it is replaced by the contents of the macro.

e.g.; define KEY_ESC 27

40. #Define:

#define is a macro and it is used widely in C whenever required. All #define values get replaced with actual values after compilation. It is used in program to clarity and ease of maintenance.

41. Putchar:

Putchar function displays a single character on the screen.

42. The time.h (Header file):

The time.h header defines four variable types, two macro and various functions for manipulating date and time.

43. Time:

Time functions refer to a group of functions in the standard library of the C programming language implementing date and time manipulation operations.

44. Sprintf:

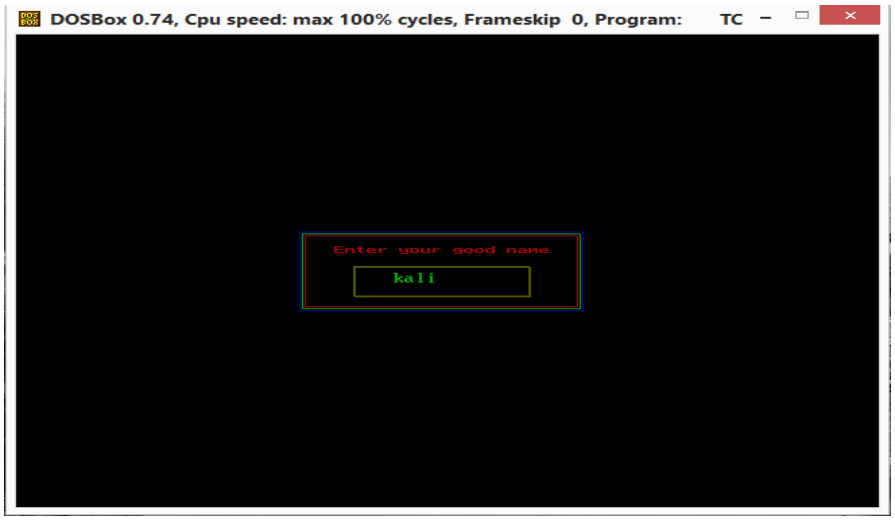
This is the string that contains the text to be printed. It can also be used to contain format tags.

45. Far:

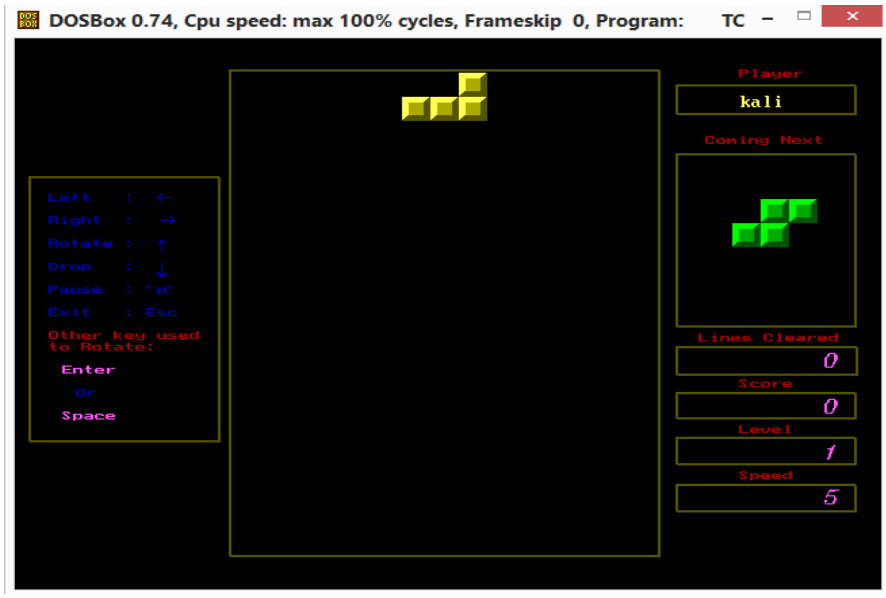
A Far function can be outside the 64KB segment most functions are force into an inadequate space into for a small-code model. The pointer which can point or access whole the residence memory of RAM i.e. which can access all 16 segments is known as far pointer. Far was a 32-bit pointer consisting of a 16-bit 'selector' that indirectly determined the start address of the memory region, and a 16-bit offset into that region.

Opening Screen of Tetris (Falling Blocks) Preview:

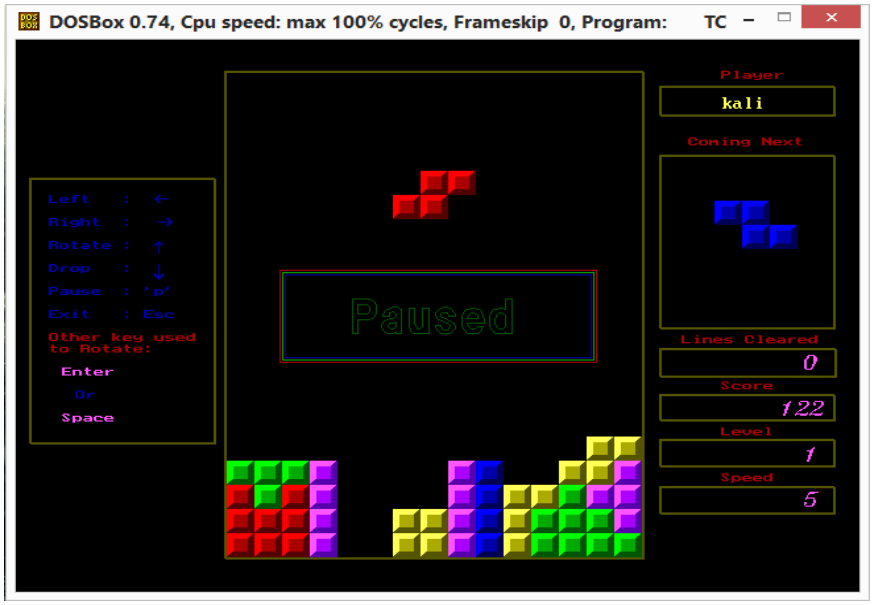
Player Name Input Screen:



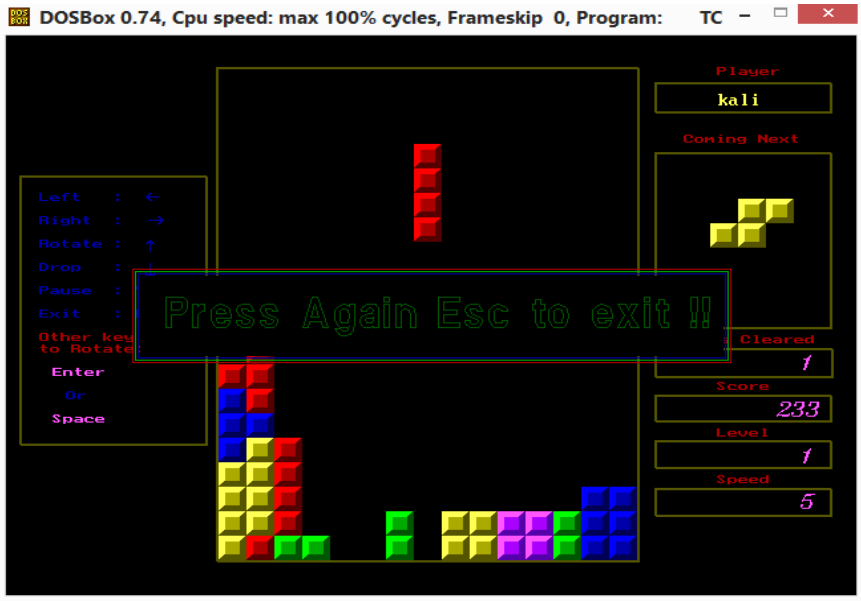
Falling Block Position and Playing Platform:



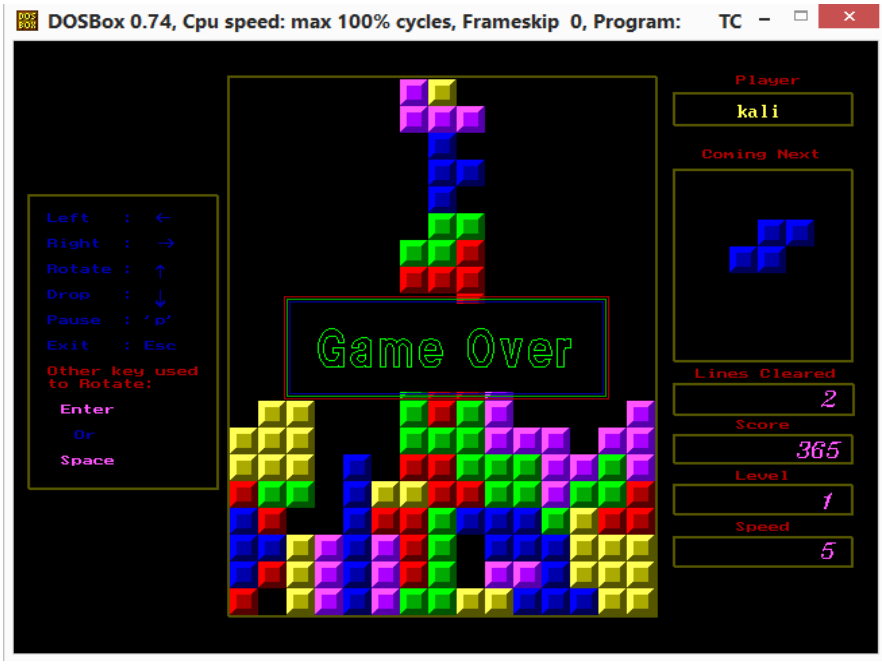
Pause Screen:



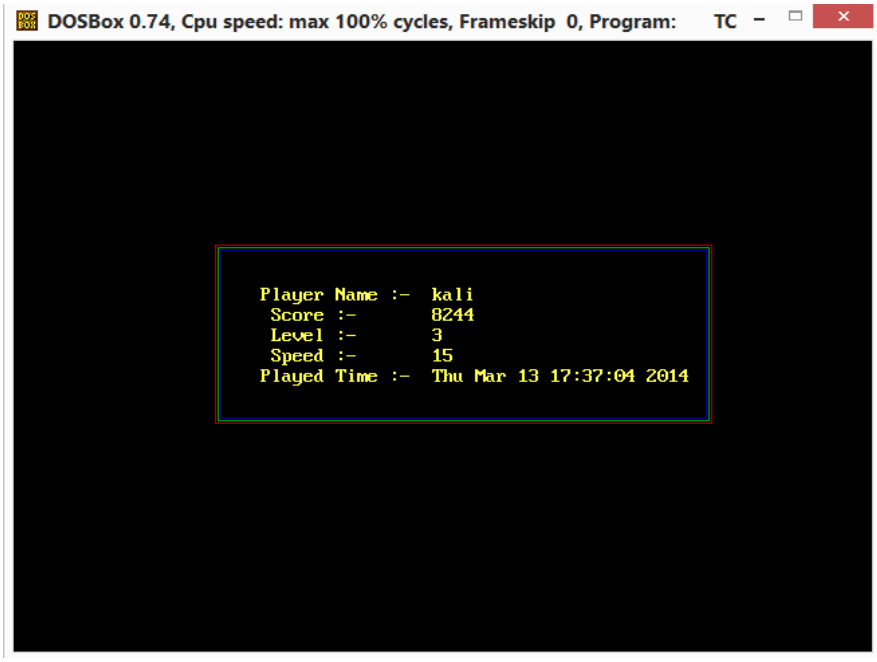
Exit Screen:



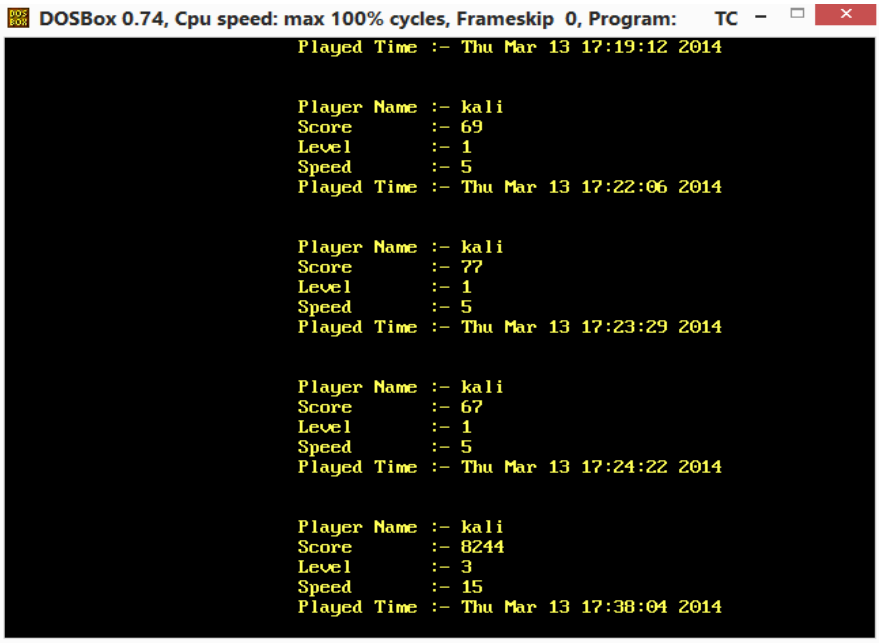
Game over Screen:



Display Player Name and Score with time and day Screen:



Display Player Previous Record Screen:



Approach to Tetris Game (Falling Blocks):

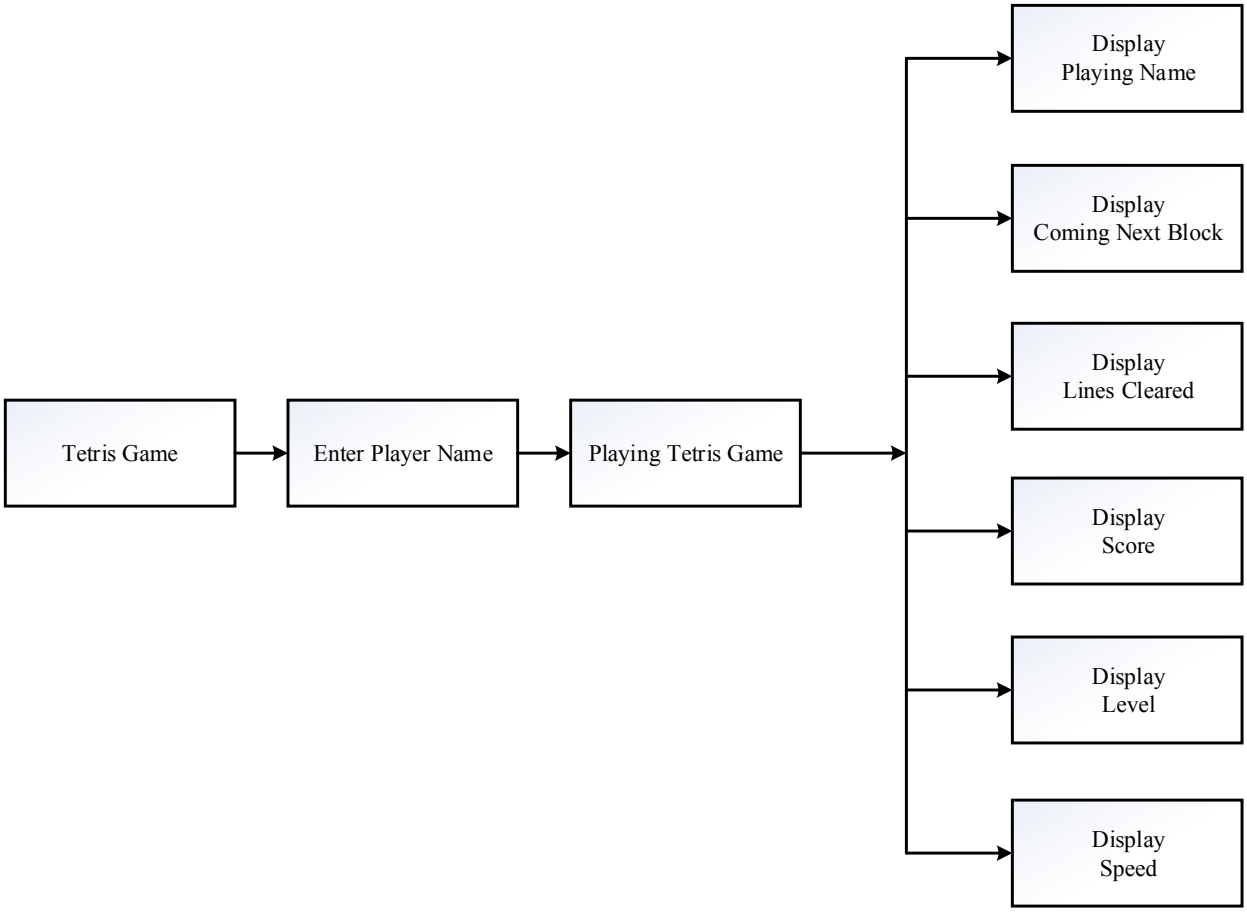


Figure : Approach to Tetris Game (Falling Blocks)

Algorithm:

1. Main():

As the program is run, it starts executing from main function.Initially, graphics driver & graphics mode are initialize. Since the global variable & symbolic constant could not be initialized in header file required global variables & symbolic constants were initialized here. Many functions like randomize(), moveblock(), assignshape(), displaynextshape(), detectcollion() ,messagebox , rotateblock() & many more functions are called here during program execution. After calling a series of function, the main function is called again. So the program terminates from the main function in this project.

Algorithm for function main()
Step 1: Start
Step 2: Initialize globally declared variables, symbolic constants & graphics driver & graphics mode
Step 3: Declare int x, y, char Key, ScanCode, ret,int Counter=0
Step 4: Start the graphics system, Call loading() function & clearviewport() in-built function to clear the graphics screen
Step 5: Call randomize() function to initializes the random number generator with a random value
Step 6: Call cleardevice ()function, InitPalette() function, InitScreenLayout() function & GetImages() function
Step 7: Call cleardevice() function, EnterBox() function, paused the program execution for 0.5 sec &call cleardevice() function
Step 8: Call AssignShape(GetRandomShape(), GetRandomColor())function to display falling block shape
Step 9: Call NextShape=GetRandomShape() function to get next shape & NextColor=GetRandomColor() function to get next shape color
Step 10: Call DisplayInterface() function to display user interface, DisplayNextShape() function to display next shape
Step 11: Call MoveBlock(LEFT)function
Step 12: If (kbhit()) getch() then goto Step 13: to see if a keystroke is currently available (empty the keyboard buffer)
Step 13: If (!Quit && !Gameover) then Step 13.1:
Step 13.1: If(++Counter >= Speed) then Counter=0
Step 13.1.1: Call MoveBlock(DOWN) function
Step 13.1.2: If(kbhit()) then goto Step 13.1.3:
Step 13.1.3: Key = getch()
Step 13.1.4: If(Key =0) then goto Step 13.1.3:
Step 13.1.4.1: ScanCode = getch()
Step 13.1.4.2: If(ScanCode = KEY_UP) then goto step 13.1.4.2.1:
Step 13.1.4.2.1: Call RotateBlock() function
Step 13.1.4.3: Else if(ScanCode == KEY_LEFT)
Step 13.1.4.3.1: Return=MoveBlock(LEFT) function
Step 13.1.4.4: Else if(ScanCode == KEY_RIGHT)
Step 13.1.4.4.1: Return=MoveBlock(RIGHT) function
Step 13.1.4.5: Else if(ScanCode = KEY_DOWN)

Step 13.1.4.5.1: Score++ increament score by once (+1)
Step 13.1.4.5.2: Call function PrintScore() function, MoveBlock(DOWN) function
Step 13.1.4.6: Return = 0
Step 13.1.5: Else if(Key = KEY_ENTER or Key = KEY_SPACE)
Step 13.1.5.1: RotateBlock() function
Step 13.1.6: Else if(Key = 'P' or Key = 'p')
Step 13.1.6.1: Call MessageBox(" Paused",240,3) function
Step 13.1.6.2: If (kbhit()) getch() to clear the keyboard buffer
Step 13.1.6.3: for(x=0; x<COLS; x++)
Step 13.1.6.3.1: for(y=0; y<ROWS; y++)
Step 13.1.6.3.2: PreviousScreenLayout[x][y] = -1 to Clear the previous screen layout to refresh whole screen
Step 13.1.6.4: Call UpdateScreen() function
Step 13.1.7: Else if(Key = KEY_ESC)
Step 13.1.7.1: Call MessageBox("Press Again Esc to exit !!",450,2) function & Declare Quit = 1
Step 13.1.7.2: Call record() function & Exit from if statement
Step 13.1.8: With a call to delay, current program is suspended from execution for 0.005sec
Step 14: If (GameOver)
Step 14.1: Call DisplayBlock(6,0) function, GameOverSound() function, ShowGameOver() function
Step 14.2: Call Cleardevice() function, MessageBox("Better Luck For Next Time !!",530,2) function
Step 14.3: Call cleardevice() function, DisplayPlayerDetail() function, record() function
Step 15: Call restorecrtmode() function to Restores screen mode to pre-initgraph setting
Step 16: End

2. Loading()
Algorithm for function loading
Step 1: Declare variable int i,j=28
Step 2: Call function cleardevice()
Step 3: Gotoxy(33,12) to moves cursor to point (33,12)
Step 4: Display Loading.....
Step 5: For (i=1;i<=20;i++)
Step 5.1: Gotoxy(j+i,13)
Step 5.2: Display character having ASCII value 177) & hold screen for 0.5sec
Step : End

3. Brick()
Algorithm for function brick
Step 1: Start
Step 2: Declare variables as int utri[6] = {x,y, x+20,y, x,y+20 }
int ltri[6] = { x+20,y+20, x+20,y, x,y+20 }
Step 3: If colour is blue
Step 3.1: Setcolor(BLUEBR), Setfillstyle(1,BLUEBR), Fillpoly(3, utri)
Step 3.2: Setcolor(BLUEDR), setfillstyle(1,BLUEDR), fillpoly(3, ltri)
Step 3.3: Setcolor(BLUE), setfillstyle(1,BLUE), bar(x+5, y+5, x+15, y+15) then Out of if statement
Step 4: Repeat steps 3.1 to 3.3 if colour is red, green, purple or yellow

4. InitPalette()
Algorithm for function initpalette() to make required color
Step 1: Start
Step 2: For blue color Setpalette(BLUE,1), setpalette(BLUEDR,8) ,setpalette(BLUEBR,9) changes colornum entry in the palette to color
Step 3: Repeat step 2 for green, yellow ,red, purple

5. GetImages()
Algorithm for function getimage() to make a required image blocks, score, level,..etc
Step 1: Start
Step 2: CallfunctionBrick(100,100,BLUE),Brick(130,100,RED),Brick(160,100,GREEN), Brick(190,100,YELLOW), Brick(220,100,PURPLE)
Step 3: Initialize bmpBlueBrick = malloc(imagesize(0,0,20,20)),
bmpRedBrick = malloc(imagesize(0,0,20,20)),
bmpGreenBrick = malloc(imagesize(0,0,20,20)),
bmpYellowBrick = malloc(imagesize(0,0,20,20)),
bmpPurpleBrick = malloc(imagesize(0,0,20,20)),
bmpBlankBrick = malloc(imagesize(0,0,20,20)),
bmpScore = malloc(imagesize(501,51,607,69)),
bmpSpeed = malloc(imagesize(501,51,607,69)),
bmpLevel = malloc(imagesize(501,51,607,69))
bmpLinesCleared = malloc(imagesize(501,51,607,69)),
bmpScreen = malloc(imagesize(0,0,640,480))

Step 4 : getimage(100,100,120,120,bmpBlueBrick),
getimage(130,100,150,120,bmpRedBrick),
getimage(160,100,180,120,bmpGreenBrick),
getimage(190,100,210,120,bmpYellowBrick),
getimage(220,100,240,120,bmpPurpleBrick),
getimage(0,0,20,20,bmpBlankBrick),
getimage(501,51,607,69,bmpScore),
getimage(501,51,607,69,bmpSpeed),
getimage(501,51,607,69,bmpLevel),
getimage(501,51,607,69,bmpLinesCleared),

6.DisplayInterface()
Algorithm for function getimage() to make a user interface display
Step 1: Draw main central block, setcolor(62), rectangle(158,27,475,450),
rectangle(159,28,476,451)
Step 2: For Title setcolor(4), outtextxy(535,27,"Player") &draw rectangles
Step 3: To display player name gotoxy(68,4) puts(PlayerName)
Step 4: For Title setcolor(4), outtextxy(510,85,"Coming Next")& draw rectangles
Step 5: Repeat step 4 for Title Lines Cleared , Score ,Level, Speed, Instruction & draw
rectangles
Step 6: Display game key instruction Left : < ,
Right : >,
Rotate : ^,
Pause : 'p',
Exit : Esc or Other key
to Rotate:Enter or Space
Step 7: To display remove rows, falling block, game speed, game level & score

7.AssignShape(int Shape, int Color)
Algorithm for function AssignShape () to make a shape of blocks as I,T,L,L2,S,S2,O
Step 1: Start
Step 2: Declare variables int i,j &CurrentShape = Shape;
Step 3: If SHAPE is I then
Step 4: For(i=0; i<4; i++)
Step 4.1: for(j=0; j<4; j++)
Step 4.2: BlockMatrix[j][i] = ShapeI[i][j]*Color then out of loop
Step 5: Repeat step 4 if shape is either T,L,L2,S,S2 or o

8. GetRandomColor()
Algorithm for function Getrandomcolor() to generate random color of blocks
Step 1: Declare int Color = rand() % 5
Step 2: If Color is 0,1,2,3,4 then return BLUE,RED,GREENYELLOW, PURPLE repectively

9. GetRandomShape()
Algorithm for function Getrandomshape() to generate random shape of blocks
Step 1: Start
Step 2:Declare int Shape = rand() % 7
Step 3: If Shape is 0,1,2,3,4,5,6 then return
SHAPE_I,SHAPE_L,SHAPE_T,SHAPE_O,SHAPE_S,SHAPE_S2,SHAPE_L2
repectively

10. MoveBlock(int Direction)
Algorithm for function MoveBlock() to make direction of blocks
Step 1: Start
Step 2: If Direction is left
Step 2.1: If(call function DetectCollision(LEFT)) then return 1
Step 2.2: Call function DisplayBlock(--BlockX, Blocky)
Step 3: Repeat step 2 if direction is RIGHT
Step 4: If Direction is DOWN
Step 4.1: If(call function DetectCollision(DOWN)) then call GetNextBlock() & return 1
Step 4.2: Call DisplayBlock(BlockX, ++BlockY);
Step 5: If direction is REFRESH then call DisplayBlock(BlockX, BlockY)

11. RotateBlock
Algorithm for function Rotateblock()to make a blocks can be rotated
Step 1: Start
Step 2: Declare int i, j, TempBlockMatrix[4][4]
Step 3: for(i=0; i<4; i++)
Step 3.1: for(j=0; j<4; j++)
Step 3.2: TempBlockMatrix[i][j] = BlockMatrix[i][j]
Step 4: If CurrentShape is SHAPE_O then return
Step 5: If CurrentShape is SHAPE_L,
case SHAPE_L2 ,
case SHAPE_s,
case SHAPE_S2,
case SHAPE_T,
then BlockMatrix[0][0] = TempBlockMatrix[2][0]
BlockMatrix[0][2] = TempBlockMatrix[0][0]
BlockMatrix[2][2] = TempBlockMatrix[0][2]
BlockMatrix[2][0] = TempBlockMatrix[2][2]
BlockMatrix[0][1] = TempBlockMatrix[1][0]
BlockMatrix[1][2] = TempBlockMatrix[0][1]
BlockMatrix[2][1] = TempBlockMatrix[1][2]
BlockMatrix[1][0] = TempBlockMatrix[2][1]
Step 6: If CurrentShape is SHAPE_I then
BlockMatrix[0][1] = TempBlockMatrix[1][0]
BlockMatrix[1][0] = TempBlockMatrix[0][1]
BlockMatrix[1][2] = TempBlockMatrix[2][1]
BlockMatrix[2][1] = TempBlockMatrix[1][2]
BlockMatrix[1][3] = TempBlockMatrix[3][1]
BlockMatrix[3][1] = TempBlockMatrix[1][3]
Step 7: If(call DetectCollision(REFRESH))
Step 7.1: for(i=0; i<4; i++)
Step 7.2: for(j=0; j<4; j++)
Step 7.3: BlockMatrix[i][j] = TempBlockMatrix[i][j] then return
Step 8: Call MoveBlock(REFRESH)

12.DisplayBlock(int x, int y)
Algorithm for function to display falling blocks
Step 1: Start
Step 2: Declare int i, j
Step 3: for(i=0; i<ROWS; i++)
Step 3.1: for(j=0; j<COLS; j++)
Step 3.1.1: ScreenLayout[j][i] = ScreenBackgroundLayout[j][i]
Step 4: for(i=0; i<4; i++)
Step 4.1: if((x+i)<0 or (x+i) > COLS) then skip the statement
Step 4.2: for(j=0; j<4; j++)
Step 4.2.1: if((y+j)>ROWS) then skip the statement
Step 4.2.2: if(BlockMatrix[i][j] == 0) then skip the statement
Step 4.2.3: ScreenLayout[x+i][y+j] = BlockMatrix[i][j]
Step 5: Call UpdateScreen()

13. DetectCollision(int Direction)
Algorithm for function Detectcollision to check boundry
Step 1: Declare int x,y, int Bx=BlockX, int By=BlockY
Step 2: If Direction is LEFT then --Bx & out of loop
Step 3: If Direction is RIGHT then ++Bx & out of loop
Step 4: If Direction is DOWN then By++ & out of loop
Step 5: Left Boundry check if(Bx < 0) then
Step 5.1: for(x=0; (x+Bx)<0; x++)
Step 5.1.1: for(y=0; y<4; y++)
Step 5.1.1.1: if(BlockMatrix[x][y]!=0) then return 1
Step 6: Right Boundry check if(Bx > COLS-4) then
Step 6.1: for(x=Bx+3; x>=COLS; x--)
Step 6.1.1: for(y=0; y<4; y++)
Step 6.1.1.1: if(BlockMatrix[x-Bx][y]!=0) then return 1
Step 7: Bottom boundry check for(x=0; x<4; x++)
Step 7.1: for(y=3; y>=0; y--)
Step 7.1.1: if((ScreenBackgroundLayout[Bx+x][By+y]!=BLANK &&
BlockMatrix[x][y]!=BLANK)
((By+y)>=ROWS && BlockMatrix[x][y]!=BLANK)) & then return 1
Step 7.2: return 0

14. GetNextBlock()
Algorithm for Functions GetNextBlock() for to get next block and it's color from GetRandomShape() & GetRandomColor

Step 1: Declare int x, y
Step 2: for(x=0; x<4; x++)
Step 2.1: for(y=0; y<4; y++)
Step 2.1.1: if(BlockMatrix[x][y] != BLANK)
Step 2.1.1.1: ScreenBackgroundLayout[BlockX+x][BlockY+y] =BlockMatrix[x][y]
Step 3: Call function CheckForLine() & AssignShape(NextShape, NextColor)
Step 4: Call function NextShape = GetRandomShape() & NextColor = GetRandomColor()
Step 5: Call function DisplayNextShape()
Step 6: Initialize BlockX = 7, BlockY = 0
Step 7: If(MoveBlock(LEFT)) then GameOver=1

15.DisplayNextShape()
Algorithm for Functions DisplayNextShape() for to display coming next blocks

Step 1: Initialize int NextTop = 140, NextLeft = 530, TempMatrix[4][4] ,i,j, x, y
Step 2: If NextShape is SHAPE_I then
Step 3: for(i=0; i<4; i++)
Step 3.1: for(j=0; j<4; j++)
Step 3.1.1: TempMatrix[j][i] = ShapeI[i][j]*NextColor
Step 4: Repeat step 3 if NextShape is SHAPE_T , SHAPE_L,SHAPEL2, SHAPE_O,SHAPE_S or SHAPE_S2
Step 5: for(x=0; x<4; x++)
Step 5.1: for(y=0; y<4; y++)
Step 5.1.1: If TempMatrix[x][y] is BLUE then
Step 5.1.1.1: putimage(NextLeft+x*21, NextTop+y*21, bmpBlueBrick, 0) & out of loop
Step 5.1.2: If TempMatrix[x][y] is RED,YELLOW, GREEN or PURPLE then Repeat step5.1.1 for each cases

//Functions for update screen
16. UpdateScreen()
Algorithm for Functions UpdateScreen() for update screen

Step 1: Start
Step 2: declare int x, y
Step 3: for(x=0; x<COLS; x++)
Step 3.1: for(y=0; y<ROWS; y++)
Step 3.1.1: if(PreviousScreenLayout[x][y] == ScreenLayout[x][y]) then skip
Step 3.1.2: PreviousScreenLayout[x][y] = ScreenLayout[x][y]
Step 3.1.3: If ScreenLayout[x][y]) is BLUE then
Step 3.1.3.1: putimage(LEFT+x*21, TOP+y*21, bmpBlueBrick, 0) &
Out of if statement

Step 3.1.4: If ScreenLayout[x][y]) is either RED, GREEN,YELLOW or PURPLE
Then repeat step 3.1.3.1 for each cases
Step 3.1.5:Else putimage(LEFT+x*21, TOP+y*21, bmpBlankBrick, 0)

17.CheckForLine()
Algorithm for Functions CheckForLine() for to check line remove or not
Step : Start
Step 2:Declare int RemoveLines[10], Remove,x,y,factor=1
Step 3: for(y=0; y<ROWS; y++)
Step 3.1: Remove=1
Step 3.2: for(x=0; x<COLS; x++)
Step 3.2.1: if(ScreenLayout[x][y] == BLANK) then Remove=0
Step 3.3: if(Remove) then RemoveLine(y)
Step 3.3.1: Score += LineScore*factor
Step 3.3.2: factor++ & call function PrintScore()
Step 4: If (kbhit()) getch() to Empty the keyboard buffer...

18. RemoveLine(int LineNumber)
Algorithm for Functions RemoveLine() for to remove line a complete rows
Step 1: Start
Step 2: Declare int i,j, count=0,Temp[COLS]
Step 3: for(i=0; i<COLS; i++)
Step 3.1: Temp[i]=ScreenLayout[i][LineNumber]

Step 4: for(j=0; j<6; j++)
Step 4.1: for(i=0; i<COLS; i++)
Step 4.1.1: ScreenLayout[i][LineNumber]=(count ? Temp[i] : BLANK)
Step 4.2: count = !count & call function UpdateScreen() & hold program execution for 0.05sec
Step 5: for(i=LineNumber; i>0; i--)
Step 5.1: for(j=0; j<COLS; j++)
Step 5.1.1: ScreenBackgroundLayout[j][i] = ScreenBackgroundLayout[j][i-1]
Step 6: for(j=0; j<COLS; j++)
Step 6.1: ScreenBackgroundLayout[j][0] = BLANK
Step 7: for(i=0; i<COLS; i++)
Step 7.1: for(j=0; j<ROWS; j++)
Step 7.1.1: ScreenLayout[i][j] = ScreenBackgroundLayout[i][j] &++LinesCleared
Step 8: for(i=500; i>0; i--)
Step 8.1:call function sound(i) & hold program execution for 0.05sec
Step 9: Call function nosound() & PrintLinesCleared()
Step 10: If(!(LinesCleared % 20)) then IncreaseSpeed()

19. IncreaseSpeed()
Algorithm for Functions IncreaseSpeed() for to increase falling blocks speed
Step 1: if(Speed-5 <= 0) then return 1
Step 2: Speed -= 5 & Level++
Step 3: Call function PrintSpeed() & PrintLevel() then return 0

20. DrawBox(int x1, int y1, int x2, int y2, int Color1, int Color2, int Color3)
Algorithm for DrawBox() Functions for to make a rectangle of rectangle for message box
Step 1: Start
Step 2:setlinestyle(0,0,1), setcolor(Color1) & draws three rectangle of different size

21.EnterBox()
Algorithm for Functions EnterBox() for input player name
Step 1: Start
Step 2: Draw three of rectangle with color red ,green, blue
Step 3: Display Enter your good name inside two rectangles
Step 4: Positions cursor at (36,16) in text window & gets playername from keyboard

22. PrintScore()
Algorithm for Function PrintScore() for to display or print score
Step 1: Start
Step 2: Sets the text font, the direction and the size of the characters
Step 3: Sends formatted output as Score
Step 4: Puts the bit image of bmpScore previously saved with getimage back onto the screen
Step 5: Display lpszScore

23. PrintLevel()
Algorithm for Function PrintLevel() for to display or print level
Step 1: Start
Step 2: Sets the text font, the direction and the size of the characters
Step 3: Sends formatted output as Level
Step 4: Puts the bit image of bmplevel previously saved with getimage back onto the screen
Step 5: Display lpszLevel

24. PrintSpeed()
Algorithm for Function PrintSpeed() for to display or print speed of falling block
Step 1: Start
Step 2: Sets the text font, the direction and the size of the characters
Step 3: Sends formatted output as 100-Speed
Step 4: Puts the bit image of bmpSpeed previously saved with getimage back onto the screen
Step 5: Display lpszSpeed

25.PrintLinesCleared()
Algorithm for Function PrintLinesCleared() for to count complete rows of blocks
Step 1: Start
Step 2: Sets the text font, the direction and the size of the characters
Step 3: Sends formatted output as bmpLinesCleared
Step 4: Puts the bit image of bmpLinesCleared previously saved with getimage back onto the screen
Step 5: Display lpszLinesCleared

26. ShowGameOver()
Algorithm for Function ShowGameOver () for to popup game over message box
Step 1: Start
Step 2: Call function MessageBox("Game Over",240,3) then return 0

27. MessageBox(char *Message, int Width, int Size)
Algorithm for Function MessageBox() for to make a message box
Step 1: Start
Step 2: Initialize int Color=0, TextX= 320-(Width/2)+25
Step 3: Sets the current fill pattern and fill color
Step 4: Draw **b**ar using the current fill pattern and fill color & call function DrawBoxGREENDR)
Step 5: Sets the current drawing color
Step 6: Sets the text font, the direction and the size of the characters
Step 7: Display lpszLinesCleared
Step 8: If (!kbhit()) kbhit checks to see if a keystroke is currently available
Step 9: If (Color++=0) then setcolor(GREENBR)
Step 10: If (Color++=1) then setcolor(GREEN)
Step 11: If (Color++=2) then setcolor(GREENdR)
Step 12: If (Color++=3) then setcolor(GREEN)
Step 13: Display Message & Suspends execution for 80 milliseconds
Step 14: if(Color>3) then Color=0
Step 15: setcolor(GREEN) & return the last key pressed form the keyboard buffer

28. GameOverSound()
Algorithm for Function GameOverSound() for to make sound for gameover
Step 1: Start
Step 2: Call function sound(600) & Suspends execution for 700 milliseconds
Step 3: Call function nosound()
Step 4: Repeat step 2 & 3 for different sound & time

29.DisplayPlayerDetail()
Step 1: Start
Step 2: set RED ,GREEN, BLUE color & draw rectangles
Step 3: Display Player Name & input PlayerName
Step 3: Positions cursor at diffrent point in text window & display Score, Level, Speed , 100-Speed

30.record()
Algorithm for Function record() for to keep database record of player
Step 1: Start
Step 2: Declare char cha,c & FILE *info
Step 3: open file record.txt in appending mode & Call in-built function cleardevice()
Step 4: Get systemtime store in mytime variable
Step 5: Write Player Name, Score, Level, Speed, Played Time
Step 6: Close info file & Call in-built function cleardevice()
Step 7: Call funtion cha = MessageBox("Wanna see past records press 'y'",580,2)
Step 8: Call in-built function cleardevice()
Step 9: if(cha=='y' or cha=='Y') then open file record.txt in reading mode
Step 10: do
Step 10.1: Read a single character from file & display it in screen
Step 10.2: If (c!=EOF) then goto step 10
Step 11: Else if(cha=='n' or cha=='N')
Step 11.1: Call function MessageBox("Thank You For Playing !!",450,2)
Step 11.2: End the program
Step 12: Call in-built function cleardevice()
Step 13: Call function MessageBox("Thank You For Playing !!",450,2) & in-built function cleardevice()
Step 14: Close file
Step 14: End

Source Code:

```
#include <graphics.h>  // to used for graphical output
#include <conio.h>   // to used for console input and output
#include <stdio.h>   // to used for standard input and output
#include <stdlib.h>  // to used for standard library function
#include <dos.h>   // to used for delay as time in millisecond delay
#include <time.h>  // to used for time

//Defines a macro for default value of color
#define BLUE 1
#define BLUEDR 2
#define BLUEBR 3
#define RED 4
#define REDDR 5
#define REDBR 6
#define GREEN 7
#define GREENDR 8
#define GREENBR 9
#define PURPLE 10
#define PURPLEDR 11
#define PURPLEBR 12
#define YELLOW 13
#define YELLOWDR 14
#define YELLOWBR 15
#define BLANK 0
#define REFRESH 0
#define ROWS 20
#define COLS 15

//Defines a macro for Shapes
#define SHAPE_I  100
#define SHAPE_T  101
#define SHAPE_L  102
#define SHAPE_L2 103
#define SHAPE_O  104
#define SHAPE_S  105
#define SHAPE_S2 106

//Defines a macro for Directions (fixed screen for falling block)
#define RIGHT 201
#define DOWN  203
#define TOP   30
#define LEFT  160

//Defines a macro for Keys
#define KEY_UP 72
```

```
#define KEY_DOWN 80
#define KEY_LEFT 75
#define KEY_RIGHT 77
#define KEY_ESC 27
#define KEY_ENTER 13
#define KEY_SPACE 32

//To Declare User Define Functions
void Brick(int x, int y, char Color);  // to make a brick
void InitPalette();  // to initialize palette
void InitScreenLayout(); // to initialize screen layout
void GetImages();    // Draws images and saves them in memory for later use
void DisplayInterface();  // to display user interface or area of games required
void AssignShape(int Shape, int Color); // Gives the shape to the current falling block
int  DetectCollision(int Direction);     // returns 1 when brick collides with something
int  MoveBlock(int Direction);  //Moves the falling block in the direction
void DisplayBlock(int x, int y); //for block review
void DisplayNextShape();
void GetNextBlock();
void RotateBlock();
void UpdateScreen();
void CheckForLine();
void RemoveLine(int Number);
int  GetRandomColor();
int  GetRandomShape();
void EnterBox();    // to display player name enter box
void PrintScore();  // to display game score
void PrintSpeed();  // to display falling blocks speed
void PrintLevel();  // to display game level
void PrintLinesCleared();   // to display how much cleared rows of bricks
void DrawBox(int x1, int y1, int x2, int y2, int Color1, int Color2, int Color3);
int  IncreaseSpeed();  // to increase falling blocks speed
int  ShowGameOver();   // to display game over
char  MessageBox(char *Message, int Width, int Size); // to display message as game over,
pause and exit
void GameOverSound();
void loading();
void record();
void DisplayPlayerDetail();

// Force pointer for Bitmaps color
void far *bmpBlueBrick;
void far *bmpRedBrick;
void far *bmpGreenBrick;
void far *bmpYellowBrick;
void far *bmpPurpleBrick;
```

```
void far *bmpBlankBrick;
void far *bmpScore;  // to force a display color score count no as integer
void far *bmpSpeed;  // to force a display color speed count no as integer
void far *bmpLevel;  // to force a display color level count no as integer
void far *bmpLinesCleared; // to force a display color linescleared count no as integer
void far *bmpScreen;

int BlockMatrix[4][4];      // Stores the shape of the current falling block
int ScreenLayout[COLS][ROWS];    // Will contain the layout with the falling block
int ScreenBackgroundLayout[COLS][ROWS]; //will contain the layout without the falling block
int PreviousScreenLayout[COLS][ROWS];  // Will contain the layout of the previous screen
int LinesCleared=0;   // to initialized linesCleared as zero at start
int Level=1;          // to initialized level as once at start
int LineScore=100;    // for one complete rows blocks score (15 blocks = 100 score)
int Speed = 95;
int CurrentShape;
int NextShape;
int NextColor;
int BlockX = 7;
int BlockY = 0;
int Quit=0;
int GameOver=0;
int Return=0;
char PlayerName[6];
char lpszScore[40];
char lpszSpeed[40];
char lpszLevel[40];
char lpszLinesCleared[40];
unsigned long Score=0;
time_t mytime;

//Shapes of the blocks
int ShapeI[4][4] = {
                    0,1,0,0,
                    0,1,0,0,
                    0,1,0,0,
                    0,1,0,0
          };

int ShapeT[4][4] = {
                    0,1,0,0,
                    0,1,1,0,
                    0,1,0,0,
                    0,0,0,0
          };
int ShapeL[4][4] = {
```

```
                0,0,1,0,
                1,1,1,0,
                0,0,0,0,
                0,0,0,0
        };
int ShapeL2[4][4] = {
                1,0,0,0,
                1,1,1,0,
                0,0,0,0,
                0,0,0,0
        };
int ShapeS[4][4] = {
                0,1,1,0,
                1,1,0,0,
                0,0,0,0,
                0,0,0,0
        };
int ShapeS2[4][4] = {
                1,1,0,0,
                0,1,1,0,
                0,0,0,0,
                0,0,0,0
        };
int ShapeO[4][4] = {
                1,1,0,0,
                1,1,0,0,
                0,0,0,0,
                0,0,0,0
        };

// Main Function
void main()
  {
        int gd=DETECT, gm;
        int x, y;
        char Key, ScanCode, ret;
        int Counter=0;
        initgraph(&gd, &gm, "c:\\tc\\bgi");   //To start the graphics system
        loading();
        clearviewport(); // Clears the current viewport
        randomize();  //randomize initializes the random number generator with a random value
        cleardevice(); // clear the graphics screen
        InitPalette();
        InitScreenLayout();
        GetImages();
        cleardevice();
```

```
        EnterBox();
        delay(500);
    cleardevice();
    AssignShape(GetRandomShape(), GetRandomColor()); // to display falling block shape
    NextShape=GetRandomShape();  // to get next shape
    NextColor=GetRandomColor();  // to get next shape color
    DisplayInterface(); // to display user interface
    DisplayNextShape(); // to display next shape
    MoveBlock(LEFT);
    while(kbhit()) getch();  /*kbhit checks to see if a keystroke is currently available (empty
the keyboard buffer)*/
    while (!Quit && !GameOver)
      {
        if(++Counter >= Speed)
          {
            Counter=0;
            MoveBlock(DOWN);
          }

        if(kbhit())
          {
            Key = getch();
            if(Key ==0)
                {
                  ScanCode = getch();
                  if(ScanCode == KEY_UP)
                                RotateBlock();
                  else if(ScanCode == KEY_LEFT)
                                Return=MoveBlock(LEFT);
                  else if(ScanCode == KEY_RIGHT)
                                Return=MoveBlock(RIGHT);
                  else if(ScanCode == KEY_DOWN) {
                                Score++;  // increament score by once (+1)
                                PrintScore();
                                MoveBlock(DOWN);
                }
                  Return = 0;
          }
          else if(Key == KEY_ENTER || Key == KEY_SPACE)
                        RotateBlock();
          else if(Key == 'P' || Key == 'p')
             {
                MessageBox("  Paused",240,3);
                while(kbhit()) getch();  //clear the keyboard buffer
                for(x=0; x<COLS; x++)
                        for(y=0; y<ROWS; y++)
```

```
                                        PreviousScreenLayout[x][y] = -1;   // Clear the previous
screen layout to refresh the whole screen
                                UpdateScreen();
                            }
                        else if(Key == KEY_ESC)
                            {
                                MessageBox("Press Again Esc to exit !!",450,2);
                                        Quit = 1;
                                        record();
                                        break;
                            }
                }
                        delay(5); /*With a call to delay, the current program is suspended from execution
for the time specified by the argument milliseconds*/
                }
                if(GameOver)
                  {
                        DisplayBlock(6,0);
                        GameOverSound();
                        ShowGameOver();
                        cleardevice();
                        MessageBox("Better Luck For Next Time !!",530,2);
                        cleardevice();
                        DisplayPlayerDetail();
                        record();
                  }
                restorecrtmode(); // Restores screen mode to pre-initgraph setting
 closegraph();
}

//Function for to display loading screen
void loading()
{
 int i,j=28;
 cleardevice();

 gotoxy(33,12);
 printf("Loading......");

   for (i=1;i<=20;i++)
     {
          gotoxy(j+i,13);
          printf("%c",177);
          delay(500);
     }
}
```

```
//Funtion for to make a block with color
void Brick(int x, int y, char Color)
   {
         int utri[6];
         int ltri[6];

         utri[0] = x;
         utri[1] = y;
         utri[2] = x+20;
         utri[3] = y;
         utri[4] = x;
         utri[5] = y+20;

         ltri[0] = x+20;
         ltri[1] = y+20;
         ltri[2] = x+20;
         ltri[3] = y;
         ltri[4] = x;
         ltri[5] = y+20;
         switch(Color)
          {
           case BLUE:
                  setcolor(BLUEBR);
                  setfillstyle(1,BLUEBR);
                  fillpoly(3, utri);
                  setcolor(BLUEDR);
                  setfillstyle(1,BLUEDR);
                  fillpoly(3, ltri);
                  setcolor(BLUE);
                  setfillstyle(1,BLUE);
                  bar(x+5, y+5, x+15, y+15);
                  break;
           case RED:
                  setcolor(REDBR);
                  setfillstyle(1,REDBR);
                  fillpoly(3, utri);
                  setcolor(REDDR);
                  setfillstyle(1,REDDR);
                  fillpoly(3, ltri);
                  setcolor(RED);
                  setfillstyle(1,RED);
                  bar(x+5, y+5, x+15, y+15);
                  break;
           case GREEN:
                  setcolor(GREENBR);
```

```
                setfillstyle(1,GREENBR);
                fillpoly(3, utri);
                setcolor(GREENDR);
                setfillstyle(1,GREENDR);
                fillpoly(3, ltri);
                setcolor(GREEN);
                setfillstyle(1,GREEN);
                bar(x+5, y+5, x+15, y+15);
                break;
        case PURPLE:
                setcolor(PURPLEBR);
                setfillstyle(1,PURPLEBR);
                fillpoly(3, utri);
                setcolor(PURPLEDR);
                setfillstyle(1,PURPLEDR);
                fillpoly(3, ltri);
                setcolor(PURPLE);
                setfillstyle(1,PURPLE);
                bar(x+5, y+5, x+15, y+15);
                break;
        case YELLOW:
                setcolor(YELLOWBR); /* setcolor sets the current drawing color to color, which
can range from 0 to getmaxcolor */
                setfillstyle(1,YELLOWBR); // setfillstyle sets the current fill pattern and fill color
                /* fillpoly draws the outline of a polygon using the current
                   line style and color, then fills the polygon using the
                   current fill pattern and fill color */
                fillpoly(3, utri);
                setcolor(YELLOWDR);
                setfillstyle(1,YELLOWDR);
                fillpoly(3, ltri);
                setcolor(YELLOW);
                setfillstyle(1,YELLOW);
                bar(x+5, y+5, x+15, y+15); // bar draws a filled-in, rectangular, two-dimensional
bar
                break;
        }
    }

//Funtion for to make own color values
void InitPalette()
    {
        // setpalette changes the colornum entry in the palette to color

        //Blue Colors
        setpalette(BLUE,1);  //Normal
```

```
        setpalette(BLUEDR,8);  //Dark
        setpalette(BLUEBR,9);  //Bright
        //Red Colors
        setpalette(RED,4);  //Normal
        setpalette(REDDR,32);  //Dark
        setpalette(REDBR,36);  //Bright
        //Green Colors
        setpalette(GREEN,2);  //Normal
        setpalette(GREENDR,16);  //Dark
        setpalette(GREENBR,18);  //Bright
        //Purple Colors
        setpalette(PURPLE,13);  //Normal
        setpalette(PURPLEDR,33);  //Dark
        setpalette(PURPLEBR,61);  //Bright
        //Yellow Colors
        setpalette(YELLOW,6);  //Normal
        setpalette(YELLOWDR,48);  //Dark
        setpalette(YELLOWBR,62);  //Bright
    }



//Funtion for to make a required image blocks, score, level,..etc
void GetImages()
    {

        Brick(100,100,BLUE);
        Brick(130,100,RED);
        Brick(160,100,GREEN);
        Brick(190,100,YELLOW);
        Brick(220,100,PURPLE);

        /* malloc allocates a block of size bytes from the memory heap.
          It allows a program to allocate memory explicitly as it's
          needed, and in the exact amounts needed */
        // imagesize determines the size of the memory area required to store a bit image
        bmpBlueBrick   = malloc(imagesize(0,0,20,20));
        bmpRedBrick    = malloc(imagesize(0,0,20,20));
        bmpGreenBrick  = malloc(imagesize(0,0,20,20));
        bmpYellowBrick = malloc(imagesize(0,0,20,20));
        bmpPurpleBrick = malloc(imagesize(0,0,20,20));
        bmpBlankBrick  = malloc(imagesize(0,0,20,20));
        bmpScore       = malloc(imagesize(501,51,607,69));
        bmpSpeed       = malloc(imagesize(501,51,607,69));
        bmpLevel       = malloc(imagesize(501,51,607,69));
```

```
        bmpLinesCleared= malloc(imagesize(501,51,607,69));
        bmpScreen      = malloc(imagesize(0,0,640,480));

        // getimage copies an image from the screen to memory
        getimage(100,100,120,120,bmpBlueBrick);
        getimage(130,100,150,120,bmpRedBrick);
        getimage(160,100,180,120,bmpGreenBrick);
        getimage(190,100,210,120,bmpYellowBrick);
        getimage(220,100,240,120,bmpPurpleBrick);
        getimage(0,0,20,20,bmpBlankBrick);
        getimage(501,51,607,69,bmpScore);
        getimage(501,51,607,69,bmpSpeed);
        getimage(501,51,607,69,bmpLevel);
        getimage(501,51,607,69,bmpLinesCleared);
    }

//Funtion for to make screen layout of falling blocks
void InitScreenLayout()
    {
        int x, y;
        for(x=0; x<COLS; x++)
          for(y=0; y<ROWS; y++) {
                ScreenLayout[x][y] = BLANK;
                PreviousScreenLayout[x][y] = BLANK;
                ScreenBackgroundLayout[x][y] = BLANK;
    }
}

//Funtion for to make user interface display
void DisplayInterface()
   {
    //Draw main central block
    setcolor(62);
    rectangle(158,27,475,450);
    rectangle(159,28,476,451);

    //For Title
    setcolor(4);
    outtextxy(535,27,"Player");

    //Draw rectangles
    setcolor(62);

    rectangle(488,40,620,65);
    rectangle(489,41,621,66);
```

```
//To display player name
gotoxy(68,4);
/* puts copies the null-terminated string s to the standard output
      stream Standard output device and appends a newline character */
puts(PlayerName);

//For Title
setcolor(4);
outtextxy(510,85,"Coming Next");

//Draw rectangles
setcolor(62);
rectangle(488,100,620,250);
rectangle(489,101,621,251);

 //For Title
setcolor(4);
outtextxy(505,257,"Lines Cleared");

//Draw rectangles
setcolor(62);
rectangle(488,268,621,292);
rectangle(489,269,622,293);

//For Title
setcolor(4);
outtextxy(535,297,"Score");

//Draw rectangles
setcolor(62);
rectangle(488,308,620,330);
rectangle(489,309,621,331);

//For Title
setcolor(4);
outtextxy(535,337,"Level");

//Draw rectangles
setcolor(62);
rectangle(488,347,620,370);
rectangle(489,348,621,371);

//For Title
setcolor(4);
outtextxy(535,377,"Speed");
```

```
 //Draw rectangles
 setcolor(62);
 rectangle(488,388,620,411);
 rectangle(489,389,621,412);

//For Title
 setcolor(0);
 outtextxy(30,100,"Instruction");

 //Draw rectangles
 setcolor(62);
 rectangle(10,120,150,350);
 rectangle(11,121,151,351);

  //for display game key instruction
  setcolor(1);
  outtextxy(25,135,"Left   :  < ");
  line(107,138,115,138);
  outtextxy(25,155,"Right  :   >");
  line(107,158,116,158);
  outtextxy(25,175,"Rotate :  ^");
  line(108,175,108,185);
  outtextxy(25,195,"Drop   :");
  line(108,195,108,205);
  setlinestyle(0,0,3);
  line(105,204,108, 207);
  line(111,204,108, 207);
  outtextxy(25,215,"Pause  : 'p' ");
  outtextxy(25,235,"Exit   : Esc");
  setcolor(4);
  outtextxy(25,255,"Other key used");
  outtextxy(25,265,"to Rotate:");
  setcolor(12);
  outtextxy(35,285,"Enter");
  setcolor(1);
  outtextxy(45,305,"Or");
  setcolor(12);
  outtextxy(35,325,"Space");

 //To display remove Or delete one rows or more than rows of blocks
 PrintLinesCleared();
 //To display falling block Or game speed
 PrintSpeed();
 //To display game level
 PrintLevel();
 //To display score
```

```
    PrintScore();
}


//Functions for to make a shape of blocks as I,T,L,L2,S,S2,O
void AssignShape(int Shape, int Color)
 {
  int i,j;
  CurrentShape = Shape;
  switch(Shape)
    {
        case SHAPE_I:
                for(i=0; i<4; i++)
                  for(j=0; j<4; j++)
                        BlockMatrix[j][i] = ShapeI[i][j]*Color;
                break;
        case SHAPE_T:
                for(i=0; i<4; i++)
                  for(j=0; j<4; j++)
                        BlockMatrix[j][i] = ShapeT[i][j]*Color;
                break;
        case SHAPE_L:
                for(i=0; i<4; i++)
                  for(j=0; j<4; j++)
                        BlockMatrix[j][i] = ShapeL[i][j]*Color;
                break;
        case SHAPE_L2:
                for(i=0; i<4; i++)
                  for(j=0; j<4; j++)
                        BlockMatrix[j][i] = ShapeL2[i][j]*Color;
                break;
        case SHAPE_O:
                for(i=0; i<4; i++)
                  for(j=0; j<4; j++)
                        BlockMatrix[j][i] = ShapeO[i][j]*Color;
                break;
        case SHAPE_S:
                for(i=0; i<4; i++)
                  for(j=0; j<4; j++)
                        BlockMatrix[j][i] = ShapeS[i][j]*Color;
                break;
        case SHAPE_S2:
                for(i=0; i<4; i++)
                  for(j=0; j<4; j++)
                        BlockMatrix[j][i] = ShapeS2[i][j]*Color;
                break;
```

```
        default:
                break;
    }
  }

//Functions for to generate random color of blocks
int GetRandomColor()
  {
   int Color = rand() % 5;
   switch(Color)
     {
        case 0 :
          return BLUE;
        case 1:
          return RED;
        case 2:
          return GREEN;
        case 3:
          return YELLOW;
        case 4:
          return PURPLE;
     }
        return 0;
  }

//Functions for to generate random shape of blocks
int GetRandomShape()
 {
  /* rand uses a multiplicative congruential random number generator
  with period 232 to return successive pseudo-random numbers in
  the range 0 to RAND_MAX */
  int Shape = rand() % 7;
  switch(Shape)
    {
        case 0 :
          return SHAPE_I;
        case 1:
          return SHAPE_L;
        case 2:
          return SHAPE_T;
        case 3:
          return SHAPE_O;
        case 4:
          return SHAPE_S;
        case 5:
          return SHAPE_S2;
```

```
        case 6:
          return SHAPE_L2;
    }
        return 0;
  }

//Functions for to make area of screen for falling blocks
int MoveBlock(int Direction)
{
  switch(Direction)
    {
        case LEFT:
                if(DetectCollision(LEFT))
                        return 1;
                DisplayBlock(--BlockX, BlockY);
                break;
        case RIGHT:
                if(DetectCollision(RIGHT))
                        return 1;
                DisplayBlock(++BlockX, BlockY);
                break;
        case DOWN:
                if(DetectCollision(DOWN)) {
                        GetNextBlock();
                        return 1;
                }
                DisplayBlock(BlockX, ++BlockY);
                break;
        case REFRESH:
                DisplayBlock(BlockX, BlockY);
                break;
    }
        return 0;
}

//Functions for to make a blocks can be rotated
void RotateBlock()
 {
    // Code to rotate the falling block
    int i, j;
    int TempBlockMatrix[4][4];
    for(i=0; i<4; i++)
            for(j=0; j<4; j++)
                    TempBlockMatrix[i][j] = BlockMatrix[i][j];
    switch(CurrentShape)
    {
```

```
        case SHAPE_O:
                return;
        case SHAPE_L:
        case SHAPE_L2:
        case SHAPE_S:
        case SHAPE_S2:
        case SHAPE_T:
                BlockMatrix[0][0] = TempBlockMatrix[2][0];
                BlockMatrix[0][2] = TempBlockMatrix[0][0];
                BlockMatrix[2][2] = TempBlockMatrix[0][2];
                BlockMatrix[2][0] = TempBlockMatrix[2][2];

                BlockMatrix[0][1] = TempBlockMatrix[1][0];
                BlockMatrix[1][2] = TempBlockMatrix[0][1];
                BlockMatrix[2][1] = TempBlockMatrix[1][2];
                BlockMatrix[1][0] = TempBlockMatrix[2][1];
                break;
        case SHAPE_I:
                BlockMatrix[0][1] = TempBlockMatrix[1][0];
                BlockMatrix[1][0] = TempBlockMatrix[0][1];

                BlockMatrix[1][2] = TempBlockMatrix[2][1];
                BlockMatrix[2][1] = TempBlockMatrix[1][2];

                BlockMatrix[1][3] = TempBlockMatrix[3][1];
                BlockMatrix[3][1] = TempBlockMatrix[1][3];
                break;
    }
        if(DetectCollision(REFRESH))
          {
                for(i=0; i<4; i++)
                        for(j=0; j<4; j++)
                          BlockMatrix[i][j] = TempBlockMatrix[i][j];
                return;
          }
        MoveBlock(REFRESH);
}

//Functions for to display falling blocks
void DisplayBlock(int x, int y)
{
        int i, j;
        for(i=0; i<ROWS; i++)
          for(j=0; j<COLS; j++)
                  ScreenLayout[j][i] = ScreenBackgroundLayout[j][i];
```

```
        for(i=0; i<4; i++)
         {
          if((x+i)<0 || (x+i) > COLS)
                  continue;
          for(j=0; j<4; j++)
            {
                  if((y+j)>ROWS)
                          continue;
                  if(BlockMatrix[i][j] == 0)
                          continue;
                  ScreenLayout[x+i][y+j] = BlockMatrix[i][j];
            }
         }
        UpdateScreen();
}

//Functions for to check boundry
int DetectCollision(int Direction)
 {
   int x,y;
   int Bx=BlockX;
   int By=BlockY;
   switch(Direction)
    {
        case LEFT:
                --Bx;
                break;
        case RIGHT:
                ++Bx;
                break;
        case DOWN:
                By++;
                break;
        default:
                break;
    }

        // Left Boundry check
        if(Bx < 0)
          {
            for(x=0; (x+Bx)<0; x++)
                  for(y=0; y<4; y++)
                          if(BlockMatrix[x][y]!=0)
                                  return 1;
          }
        // Right Boundry check
```

```
        if(Bx > COLS-4)
        {
          for(x=Bx+3; x>=COLS; x--)
                for(y=0; y<4; y++)
                        if(BlockMatrix[x-Bx][y]!=0)
                                return 1;
        }
        // Bottom boundry check
        for(x=0; x<4; x++)
                for(y=3; y>=0; y--)
                 {
                        if((ScreenBackgroundLayout[Bx+x][By+y]!=BLANK &&
                                BlockMatrix[x][y]!=BLANK) ||
                                ((By+y)>=ROWS && BlockMatrix[x][y]!=BLANK))
                                return 1;
                 }

        return 0;
}

/*Functions for to get next block and it's color
from GetRandomShape() & GetRandomColor*/
void GetNextBlock()
{
        int x, y;
        for(x=0; x<4; x++)
          for(y=0; y<4; y++)
                if(BlockMatrix[x][y] != BLANK)
                        ScreenBackgroundLayout[BlockX+x][BlockY+y] = BlockMatrix[x][y];
        CheckForLine();
        AssignShape(NextShape, NextColor);
        NextShape = GetRandomShape();
        NextColor = GetRandomColor();
        DisplayNextShape();
        BlockX = 7;
        BlockY = 0;
        if(MoveBlock(LEFT))
          {
            GameOver=1;
          }
}

//Functions for to display coming next blocks
void DisplayNextShape()
{
  int NextTop = 140;
```

```
int NextLeft = 530;
int TempMatrix[4][4];
int i,j;
int x, y;

switch(NextShape)
  {
      case SHAPE_I:
            for(i=0; i<4; i++)
              for(j=0; j<4; j++)
                  TempMatrix[j][i] = ShapeI[i][j]*NextColor;
            break;
      case SHAPE_T:
            for(i=0; i<4; i++)
              for(j=0; j<4; j++)
                  TempMatrix[j][i] = ShapeT[i][j]*NextColor;
            break;
      case SHAPE_L:
            for(i=0; i<4; i++)
              for(j=0; j<4; j++)
                  TempMatrix[j][i] = ShapeL[i][j]*NextColor;
            break;
      case SHAPE_L2:
            for(i=0; i<4; i++)
              for(j=0; j<4; j++)
                  TempMatrix[j][i] = ShapeL2[i][j]*NextColor;
            break;
      case SHAPE_O:
            for(i=0; i<4; i++)
              for(j=0; j<4; j++)
                  TempMatrix[j][i] = ShapeO[i][j]*NextColor;
            break;
      case SHAPE_S:
            for(i=0; i<4; i++)
              for(j=0; j<4; j++)
                  TempMatrix[j][i] = ShapeS[i][j]*NextColor;
            break;
      case SHAPE_S2:
            for(i=0; i<4; i++)
              for(j=0; j<4; j++)
                  TempMatrix[j][i] = ShapeS2[i][j]*NextColor;
            break;
      default:
            break;
  }
```

```
        for(x=0; x<4; x++)
           for(y=0; y<4; y++)
             {
                switch(TempMatrix[x][y])
                {
                 case BLUE:
                        putimage(NextLeft+x*21, NextTop+y*21, bmpBlueBrick, 0);
                        break;
                 case RED:
                        putimage(NextLeft+x*21, NextTop+y*21, bmpRedBrick, 0);
                        break;
                 case GREEN:
                        putimage(NextLeft+x*21, NextTop+y*21, bmpGreenBrick, 0);
                        break;
                 case YELLOW:
                        putimage(NextLeft+x*21, NextTop+y*21, bmpYellowBrick, 0);
                        break;
                 case PURPLE:
                        putimage(NextLeft+x*21, NextTop+y*21, bmpPurpleBrick, 0);
                        break;
                 default:
                        putimage(NextLeft+x*21, NextTop+y*21, bmpBlankBrick, 0);
                }
             }
}

//Functions for update screen
void UpdateScreen()
  {
        int x, y;
        for(x=0; x<COLS; x++)
          for(y=0; y<ROWS; y++)
             {
                 if(PreviousScreenLayout[x][y] == ScreenLayout[x][y])
                        continue;   //No need to draw the same image again
                 PreviousScreenLayout[x][y] = ScreenLayout[x][y];
                switch(ScreenLayout[x][y])
                {
                 case BLUE:
                 /*  putimage puts the bit image previously saved with getimage back onto the
                    screen, with the upper left corner of the image placed at (left,top) */
                        putimage(LEFT+x*21, TOP+y*21, bmpBlueBrick, 0);
                        break;
                 case RED:
                        putimage(LEFT+x*21, TOP+y*21, bmpRedBrick, 0);
                        break;
```

```
                case GREEN:
                        putimage(LEFT+x*21, TOP+y*21, bmpGreenBrick, 0);
                        break;
                case YELLOW:
                        putimage(LEFT+x*21, TOP+y*21, bmpYellowBrick, 0);
                        break;
                case PURPLE:
                        putimage(LEFT+x*21, TOP+y*21, bmpPurpleBrick, 0);
                        break;
                default:
                        putimage(LEFT+x*21, TOP+y*21, bmpBlankBrick, 0);
                }
        }
}

//Functions for to check line remove or not
void CheckForLine()
  {
        int RemoveLines[10];
        int Remove;
        int x,y;
        int factor=1;
        for(y=0; y<ROWS; y++)
         {
          Remove=1;
          for(x=0; x<COLS; x++)
            {
                if(ScreenLayout[x][y] == BLANK)
                        Remove=0;
            }
          if(Remove)
            {
                RemoveLine(y);
                Score += LineScore*factor;
                factor++;
                PrintScore();
            }
         }
        while (kbhit()) getch(); //Empty the keyboard buffer...
}

//Functions for to remove line a complete rows
void RemoveLine(int LineNumber)
  {
        int i,j;
        int count=0;
```

```
        int Temp[COLS];
        for(i=0; i<COLS; i++)
         {
           Temp[i]=ScreenLayout[i][LineNumber];
         }
        for(j=0; j<6; j++)
         {
                for(i=0; i<COLS; i++) {
                  ScreenLayout[i][LineNumber]=(count ? Temp[i] : BLANK);
         }
                count = !count;
                UpdateScreen();
                delay(50);
        }
        for(i=LineNumber; i>0; i--)
            for(j=0; j<COLS; j++)
            {
                    ScreenBackgroundLayout[j][i] = ScreenBackgroundLayout[j][i-1];
            }
        for(j=0; j<COLS; j++)
           ScreenBackgroundLayout[j][0] = BLANK;

        for(i=0; i<COLS; i++)
           for(j=0; j<ROWS; j++)
                    ScreenLayout[i][j] = ScreenBackgroundLayout[i][j];
        ++LinesCleared;
        //printf("boooom.....\n");
        for(i=500; i>0; i--)
         { sound(i);
           delay(1); }
        nosound();

        PrintLinesCleared();
        if(!(LinesCleared % 20))
           IncreaseSpeed();
}

//Functions for to increase falling blocks speed
int IncreaseSpeed()
  {
        if(Speed-5 <= 0)
                return 1;
        Speed -= 5;  //increase the speed... lower the number, higher the speed
        Level++;
        PrintSpeed();
        PrintLevel();
```

```
        return 0;
 }

//Functions for to make a rectangle for message box
void DrawBox(int x1, int y1, int x2, int y2, int Color1, int Color2, int Color3)
 {
        setlinestyle(0,0,1);  //Sets the current line style and width or pattern
        setcolor(Color1);
        rectangle(x1,y1,x2,y2);  //Draws a rectangle (graphics mode)
        setcolor(Color2);
        rectangle(x1+2,y1+2,x2-2,y2-2);
        setcolor(Color3);
        rectangle(x1+4,y1+4,x2-4,y2-4);
 }

//Functions for input player name
void EnterBox()
  {
        setcolor(BLUE);
        rectangle(210,200,420,280);
        setcolor(GREEN);
        rectangle(212,202,418,278);
        setcolor(RED);
        rectangle(214,204,416,276);
        setcolor(4);
        outtextxy(236,215,"Enter your good name");

        //Draw a lines for the top and left side
        setcolor(62);
        /* line draws a line from (x1, y1) to (x2, y2) using the current
           color, line style, and thickness. It does not update the current
           position (CP). */
        rectangle(250,235,380,265);
        rectangle(251,236,381,266);

        gotoxy(36,16);  // Positions cursor in text window
        gets(PlayerName); //gets gets a string from Standard input device
 }

//Function for to display or print score
void PrintScore()
 {
        /* settextstyle sets the text font, the direction in which text is
           displayed, and the size of the characters */
        settextstyle(7,0,1);
        //sprintf  sends formatted output to a string
```

```
        sprintf(lpszScore, "%11ld", Score);
        /* putimage puts the bit image previously saved with getimage back
           onto the screen, with the upper left corner of the image placed
           at (left,top) */
        putimage(510,311, bmpScore,0);
        /* outtextxy display a text string, using the current justification
           settings and the current font, direction, and size */
        outtextxy(508,308,lpszScore);
 }

//Function for to display or print level
void PrintLevel()
 {
        settextstyle(7,0,1);
        sprintf(lpszLevel, "%11d", Level);
        putimage(510,351, bmpLevel,0);
        outtextxy(508,348,lpszLevel);
 }

//Function for to display or print speed of falling block
void PrintSpeed()
 {
        settextstyle(7,0,1);
        sprintf(lpszSpeed, "%11d", 100-Speed);
        putimage(510,390, bmpSpeed,0);
        outtextxy(508,387,lpszSpeed);
 }

//Function for to count complete rows of blocks
void PrintLinesCleared()
  {
        settextstyle(7,0,1);
        sprintf(lpszLinesCleared, "%11d", LinesCleared);
        putimage(510,271, bmpLinesCleared,0);
        outtextxy(508,268,lpszLinesCleared);
  }

//Function for to popup game over message box
int ShowGameOver()
 {
        MessageBox("Game Over",240,3);
        return 0;
 }
```

```
//Function for to make a message box
char  MessageBox(char *Message, int Width, int Size)
  {
        //Width defaults to 240
        int Color=0;
        int TextX= 320-(Width/2)+25;
        //setfillstyle sets the current fill pattern and fill color
        setfillstyle(0, YELLOWDR);
        /* The bar is filled using the current fill pattern and fill color.
           bar does not outline the bar. */
        bar(320-(Width/2)+6,206,320+(Width/2)-6,274);
        DrawBox(320-(Width/2),200,320+(Width/2),280, RED, GREEN, BLUE);
        setfillstyle(10, GREENDR);
        // setcolor sets the current drawing color to color
        setcolor(GREENBR);
        settextstyle(10,0,Size);
        outtextxy(TextX,210, Message);
        // kbhit checks to see if a keystroke is currently available
        while(!kbhit())
        {
          switch(Color++)
          {
          case 0:
                        setcolor(GREENBR);
                        break;
          case 1:
                        setcolor(GREEN);
                        break;
          case 2:
                        setcolor(GREENDR);
                        break;
          case 3:
                        setcolor(GREEN);
                        break;
          }
          outtextxy(TextX,210, Message);
          // Suspends execution for interval (milliseconds)
          delay(80);
          if(Color>3)
                Color=0;
        }
        setcolor(GREEN);
        // getch gets a character from console but does not echo to the screen
        return getch();  // return the last key pressed form the keyboard buffer
}
```

```
//Function for to make sound for gameover
void GameOverSound()
  {  //printf("sa.....\n");
     sound(600);
     delay(100);
     nosound();
     //printf("re.....\n");
     sound(700);
     delay(100);
     nosound();
     //printf("ga.....\n");
     sound(800);
     delay(100);
     nosound();
     //printf("ma.....\n");
     sound(900);
     delay(100);
     nosound();
     //printf("pa.....\n");
     sound(1000);
     delay(100);
     nosound();
     //printf("dha.....\n");
     sound(1100);
     delay(100);
     nosound();
     //printf("ni.....\n");
     sound(1200);
     delay(100);
     nosound();
  }

//Function for to display player detail after game over
void DisplayPlayerDetail()
 {  setcolor(RED);
    rectangle(150,160,520,300);
    setcolor(GREEN);
    rectangle(152,162,518,298);
    setcolor(BLUE);
    rectangle(154,164,516,296);
    time(&mytime);
    gotoxy(24,13);
    printf("Player Name :-");
    gotoxy(40,13);
    puts(PlayerName);
    gotoxy(25,14);
```

```
    printf("Score :-");
    gotoxy(40,14);
    printf("%d",Score);
    gotoxy(25,15);
    printf("Level :-");
    gotoxy(40,15);
    printf("%d",Level);
    gotoxy(25,16);
    printf("Speed :-");
    gotoxy(40,16);
    printf("%d",100-Speed);
    gotoxy(24,17);
    printf("Played Time :-");
    gotoxy(40,17);
    printf("%s",ctime(&mytime));
    getch(); }

/*Function for to record player detail in record.txt file
and show it when player required to display with previous data*/
void record()
  {
    char cha,c;
    FILE *info;
    info=fopen("record.txt","a+");
    cleardevice();

    time(&mytime);
    fprintf(info,"\n\n\t\t      Player Name :- %s ",PlayerName);
    fprintf(info,"\n\t\t      Score       :- %d ",Score);
    fprintf(info,"\n\t\t      Level       :- %d ",Level);
    fprintf(info,"\n\t\t      Speed       :- %d ",100-Speed);
    fprintf(info,"\n\t\t      Played Time :- %s",ctime(&mytime));
    fclose(info);
    cleardevice();
    cha = MessageBox("Wanna see past records press 'y'",580,2);
    cleardevice();
    if(cha=='y' || cha=='Y')
    {
      info=fopen("record.txt","r");

      do
           { putchar(c=getc(info)); }
       while (c!=EOF);
    }
    else if(cha=='n' || cha=='N')
      { MessageBox("Thank You For Playing !!",450,2);
```

```
        exit(1); }
    getch();
    cleardevice();
    MessageBox("Thank You For Playing !!",450,2);
    cleardevice();
    fclose(info);
}
```

Conclusion:

In conclusion, we still want to emphasize that the program is not complete by itself. There is still a lot of room for improvement.

This is an open source program and therefore everybody is welcome to develop it. Future developers are very welcome to add their ideas to the program and improvise it.

Hopefully, all the users of this program will find it useful.

Programming tools:

Turbo C Graphics:

The approach to graphics that we use in this project relies on the existence of the **directory\TC\BGI** in the system. The last argument to the initgraph () function specifies the path to this directory, and the graphics functions expect to find various files there. One of these files is the graphics driver.

Bibliography:

- **Books:**
  - ✓ Object oriented programming in turbo C++ by Robert Lafores
  - ✓ Thinking in C++ by Sunil K Pandey
- **Internet sites:**
  - ✓ www.programmingsimplified.com
  - ✓ www.javilop.com

Annexes:

We use the following keyword character for these operations below:

S.N	Keyword Character	Operation	ASCII value
1.	KEY_UP	To rotate coming falling blocks	72
2.	KEY_DOWN	To speed up the falling blocks from top to bottom Boundry	80
3.	KEY_LEFT	To move left Boundry coming falling blocks	75
4.	KEY_RIGHT	To move right Boundry coming falling blocks	77
5.	KEY_ESC	To exit form Tetris game (falling blocks)	27
6.	KEY_ENTER	To rotate coming falling blocks	13
7.	KEY_SPACE	To rotate coming falling blocks	32
8.	Y or y	For Yes option	-
9.	N or n	For No option	-