

Online Tetris Game

Web Programming Project

By : Kanika Mathur

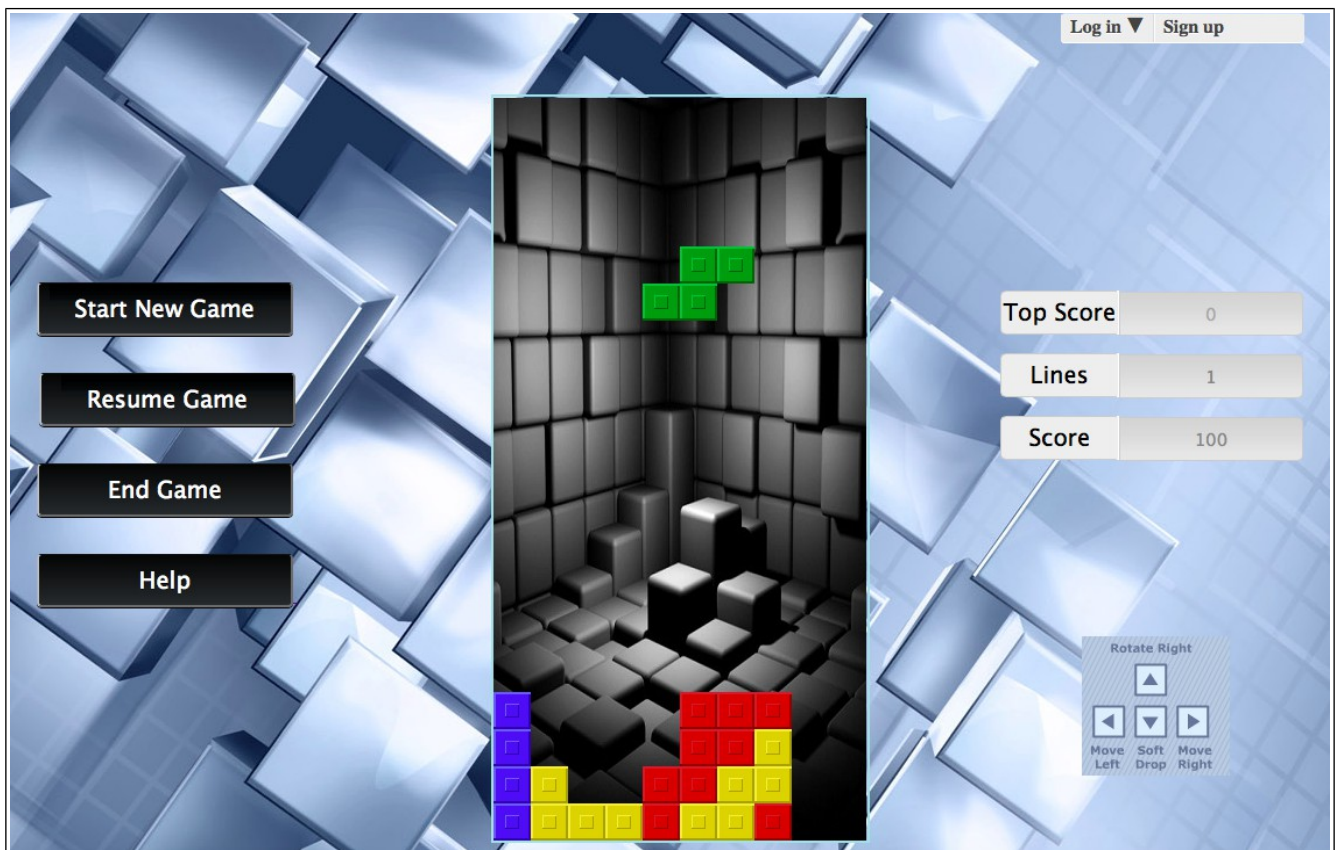


Table of Contents

Introduction.....	3
Game Components.....	4
Game Canvas.....	5
Game Pieces.....	6
Playing New Game.....	9
Pause Game.....	13
End Game.....	14
Score calculation.....	14
New user registration.....	15
Player Login.....	18
Previous Scores.....	20
Save Score.....	22
Background Music.....	24
Help Page.....	24
Database.....	25

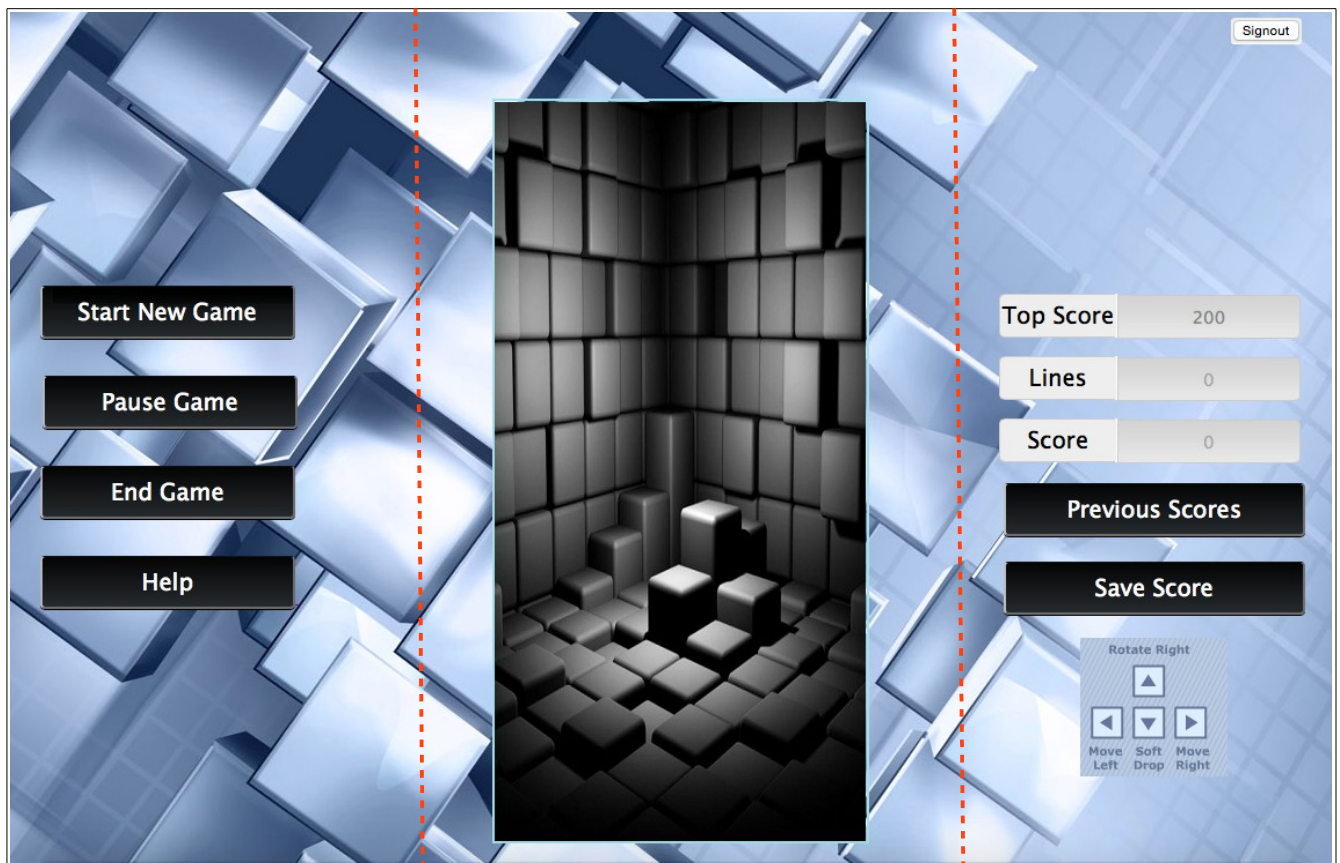
Introduction

A classic online Tetris game is developed as a part of coursework for Web Programming. It is implemented using HTML/CSS, Javascript, JSON and AJAX for front-end design and PHP and MySQL for backend features. Following are the salient features of the game project:

1. Play a new game which includes implementation of basic client side logic for block generation, block movement and rotation, line completion check and boundary and game end checks using Javascript.
2. Pause and resume an ongoing game.
3. End game.
4. Score calculation based on number of lines completed.
5. New user registration and account creation.
6. Existing user login.
7. Retrieve previous scores for a user.
8. Fetching top score among all the users who have played the game so far.
9. Save score at the end of the game.
10. Background Tetris game music.
11. Help pages.

Game Components

The game page is divided into three sections using float layout. The left pane contains game controls for starting a game, pausing and ending an ongoing game and help buttons. The center pane consists of Tetris game canvas where the actual game is played. The right pane consists of score information like Top score, Current score, Number of Lines and buttons for saving the score and viewing previous scores.



Left Game control Pane

Center Pane Game Canvas

Right Pane Score Info

Game Canvas

The game board is of size 320px X 640px which is represented as a 2D grid where each grid size is 4px X 4px.

```

<div id="tetrisboard">
  <canvas id="gameCanvas" width="320" height="640" style="border:2.5pxsolid
#B0E0E6"></canvas>
  <script>
    var canvas = document.getElementById('gameCanvas');
    var context = canvas.getContext('2d');
    var imageObj = new Image();

    imageObj.onload = function() {
      context.drawImage(imageObj, 0, 0,320,640,0,0,320,640);
    };
    imageObj.src = '../images/bg.png';
  </script>
</div>

```

Game Pieces

Tetris game consists of seven different pieces. These pieces are also represented as 2D arrays of block where a '1' represent the presence of a 4px X 4px block. Each orientation of a piece is represented by a form array.

Line Piece



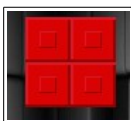
```

function Line()
{
  this.form1 = [ [1,1,1,1] ];
  this.form2 = [ [1],[1],[1],[1] ];

  this.forms = [this.form1, this.form2];
  this.currentform = 0;
  this.color=0;
  this.gridx=2;
  this.gridy = -1;
}

```

Square Piece



```
function Square()
{
    this.form1 = [ [1,1],[1,1] ];
    this.forms = [this.form1];
    this.currentform = 0;
    this.color=0;
    this.gridx=4;
    this.gridy = -2;
}
```

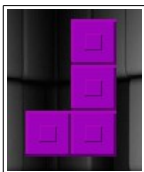
L Piece



```
function L()
{
    this.form1 =[ [1,0], [1,0], [1,1]];
    this.form2= [ [0,0,1],[1,1,1]];
    this.form3 =[ [1,1], [0,1], [0,1]];
    this.form4= [ [1,1,1],[1,0,0] ];

    this.forms = [this.form1, this.form2,
this.form3, this.form4];
    this.currentform = 0;
    this.color=0;
    this.gridx=4;
    this.gridy = -3;
}
```

Reverse L Piece



```
function ReverseL(){
    this.form1 =[ [0,1], [0,1], [1,1]];
    this.form2= [ [1,1,1],[0,0,1]];
    this.form3 =[ [1,1], [1,0], [1,0]];
    this.form4= [ [1,0,0],[1,1,1] ];

    this.forms = [this.form1, this.form2,
this.form3, this.form4];
    this.currentform = 0;
    this.color=0;
    this.gridx=4;
    this.gridy = -3;
}
```

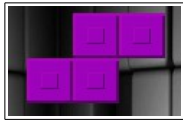
Z Piece



```
function Z()
{
    this.form1 =[ [1,1,0], [0,1,1]];
    this.form2= [ [0,1],[1,1],[1,0] ];

    this.forms = [this.form1, this.form2];
    this.currentform = 0;
    this.color=0;
    this.gridx=4;
    this.gridy = -2;
}
```

Reverse Z Piece



```
function ReverseZ(){
    this.form1= [ [0,1,1],[1,1,0]];
    this.form2 =[ [1,0], [1,1], [0,1]];

    this.forms = [this.form1, this.form2];
    this.currentform = 0;
    this.color=0;
    this.gridx=4;
    this.gridy = -2;
}
```

T Piece



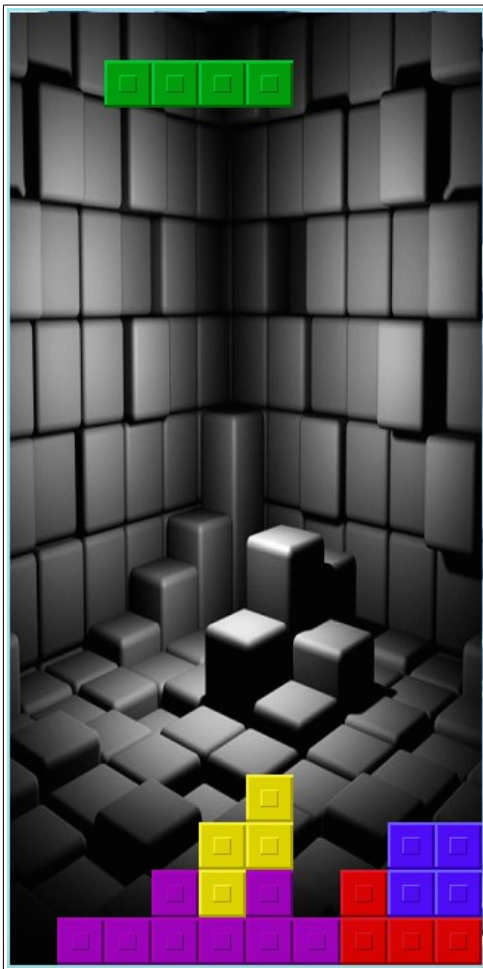
```
function T()
{
    this.form1 =[ [1,1,1], [0,1,0]];
    this.form2= [ [1,0],[1,1],[1,0] ];
    this.form3 =[ [0,1,0], [1,1,1]];
    this.form4= [ [0,1],[1,1],[0,1] ];

    this.forms = [this.form1, this.form2,
this.form3, this.form4];
}
```

```
this.currentform = 0;  
  this.color=0;  
  this.gridx=4;  
  this.gridy = -2;  
}
```

Playing New Game

When HTML page gets loaded all the background page images, game board, buttons and text boxes are loaded. Certain game parameters are also initialized to their default values. When user clicks Start New Game button, the blocks of different shapes are randomly generated and move down the canvas using animate function. The blocks can be rotated by user using the keypress functions and can be shifted to left or right. On pressing the downKey, the block falls at the faster pace. When the blocks reach the end of canvas they are controlled by boundary function to stay within the realms of Game Board. When the blocks completely fills the y axis of the board, game is over. A game over image is displayed to indicate the event.



During the game



On Game over

***Initialize()* function :**

Used to initialize 2D array representing the game board. Certain other game parameters like score and number of lines completed are also initialized to default values. window.requestAnimationFrame method is used to update the animation before repaint.

```
function initialize()
{
    var r, c;
    curLines = 0;
    prevLines = 0;
    score = 0;
    isGameOver = false;
    pauseGame = false;

    if(gameData == undefined)
    {
        gameData = new Array();

        for(r = 0; r ≤ ROWS; r++)
        {
            gameData[r] = new Array();
            for(c = 0; c < COLS; c++)
            {
                gameData[r].push(0);
            }
        }
    }
    else
    {
        for(r = 0; r < ROWS; r++)
        {
            for(c = 0; c < COLS; c++)
            {
                gameData[r][c] = 0;
            }
        }
    }

    curPiece = genBlock();

    document.getElementById("lines").value = curLines.toString();
    document.getElementById("score").value = score.toString();

    var requestAnimFrame = window.requestAnimationFrame ||
window.mozRequestAnimationFrame ||
window.webkitRequestAnimationFrame ||
window.msRequestAnimationFrame;

    window.requestAnimationFrame = requestAnimFrame;

    requestAnimationFrame(updateGameBoard);
}
```

***updateGameBoard()* function:**

This function updates the game board at regular time interval. This method in turn calls checkMove() to check if the piece is within the bounds of the board. If yes, y-coordinate of the piece is incremented. If the piece reaches the bottom of the board a new piece is generated by calling genBlock().

```
function updateGameBoard()
{
    curTime = new Date().getTime();

    if((curTime - prevTime ≥ 500) && ( pauseGame == false))
    {
        // update the game piece
        if( checkMove(curPiece.gridx, curPiece.gridy + 1,
curPiece.currentform) )
        {
            curPiece.gridy += 1;
        }
        else
        {
            copyData(curPiece);
            curPiece = genBlock();
        }

        // update time
        prevTime = curTime;
    }

    ctx.clearRect(0, 0, 320, 640);
    drawBoard();
    drawPiece(curPiece);

    if(isGameOver == false)
    {
        requestAnimationFrame(updateGameBoard);
    }
    else{
        ctx.drawImage(gameOverImg, 0, 0, 320, 640, 0, 0, 320, 640);
        endGame();
    }
}
```

***checkMove()* function:**

This function checks the x and y coordinates and current orientation of the piece and determines whether the piece can descend down or not. This function is also responsible for checking the left and right boundary checks.

```

function checkMove(xpos, ypos, newState)
{
    var result = true;
    var newx = xpos;
    var newy = ypos;

    for(var r = 0, len = curPiece.forms[newState].length; r ≤ len; r++)
    {
        for(var c = 0, len2 = curPiece.forms[newState][r].length; c < len2;
c++)
        {
            if(newx < 0 || newx ≥ COLS)
            {
                result = false;
                c = len2;
                r = len;
            }

            if(gameData[newy] != undefined && gameData[newy][newx] != 0
&& curPiece.forms[newState][r] != undefined &&
curPiece.forms[newState][r][c] != 0)
            {
                result = false;
                c = len2;
                r = len;
            }

            newx += 1;
        }

        newx = xpos;
        newy += 1;

        if(newy > ROWS)
        {
            r = len;
            result = false;
        }
    }
    return result;
}

```

***getInput()* function :**

getInput() function takes the player key strokes and based on the keyCode updates the x and y coordinates of the piece.

```

function getInput(e)
{
    if(!e) { var e = window.event; }
    e.preventDefault();
    if(isGameOver != true)
    {
        switch(e.keyCode)
        {
            case 37:
            {
                if( checkMove(curPiece.gridx - 1, curPiece.gridy, curPiece.currentform)
                {
                    curPiece.gridx--;
                }
                break;
            }
            case 39:
            {
                if( checkMove(curPiece.gridx + 1, curPiece.gridy, curPiece.currentform)
                {
                    curPiece.gridx++;
                }
                break;
            }
            case 38:
            {
                var newState = curPiece.currentform - 1;
                if(newState ≤ 0)
                    newState = curPiece.forms.length - 1;

                if( checkMove(curPiece.gridx, curPiece.gridy, newState) )
                    curPiece.currentform = newState;
            }
            break;
            case 40:
            {
                if( checkMove(curPiece.gridx, curPiece.gridy + 1, curPiece.currentform)
                {
                    curPiece.gridy++;
                }
                break;
            }
        }
    }
    else
    {
        initialize();
    }
}

```

Pause Game

Player can pause the game by clicking Pause button. A global variable called pauseGame is maintained in Javascript. Whenever this variable is true the periodic updates of game board is disabled. On clicking the Pause button the value of button toggles to Resume Game. When the game is paused the background music is also paused.

```
function pause(){
    var elem = document.getElementById("pause");
    var music = document.getElementById("audio");
    if(isGameOver == false){
        if (elem.value=="Resume Game")
        {
            music.play();
            elem.value = "Pause Game";
            pauseGame = false;
        }
        else {
            elem.value = "Resume Game";
            pauseGame = true;
            music.pause();
        }
    }
}
```

End Game

User can end the game by clicking on the end game button. A global variable isGameOver is set to true in Javascript. Also, clicking on end game button, alerts the user to signup or login to save his score before quitting the game.

```
function endGame(){
    isGameOver = true;
    var elem = document.getElementById("audio");
    elem.pause();
    elem.currentTime = 0;
    if(!
(document.getElementById("inscore").style.visibility
== "visible"))
        alert("Please Log-in or Sign up to save
score");}
```

Score calculation

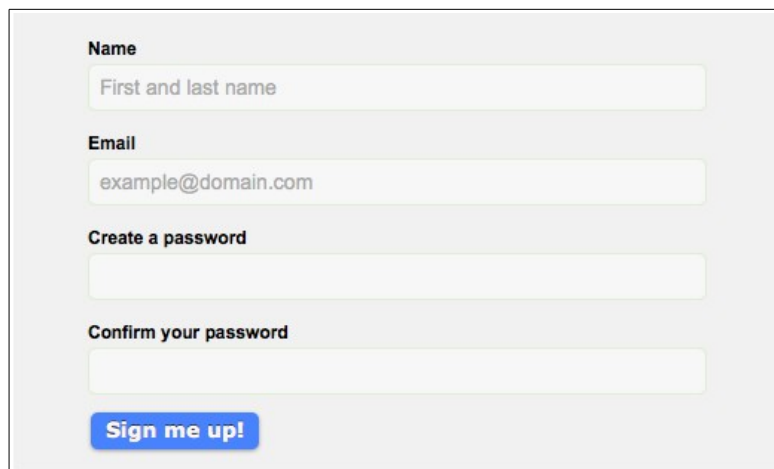
The scoring formula for the majority of Tetris products is built on the idea that more difficult line clears should be awarded more points. For example, a single line clear in Tetris Zone is worth 100 points, clearing two lines at once is worth 400 points, clearing three lines at once is worth 800 points and clearing four lines at once (known as a Tetris) is worth 1200.

```
function updateScore(){
    var diff = curLines - prevLines;
    switch(diff){
        case 0: score = score;
        break;
        case 1: score = score + 100;
```

```
break;
        case 2: score = score + 400;
        break;
        case 3: score = score + 600;
        break;
        case 4: score = score + 1200;
        break;
    }
    document.getElementById("score").value=
score.toString();
}
```

New user registration

A user can create their account by clicking on signup link on the top right corner of the page. Clicking on the sign up button opens up a new HTML page where player can enter his Name, Email Id and Password.

A screenshot of a web form titled "Sign Up Page". The form is set against a light gray background. It contains four input fields: "Name" with placeholder text "First and last name", "Email" with placeholder text "example@domain.com", "Create a password", and "Confirm your password". Below these fields is a blue button with white text that says "Sign me up!".

Name
First and last name

Email
example@domain.com

Create a password

Confirm your password

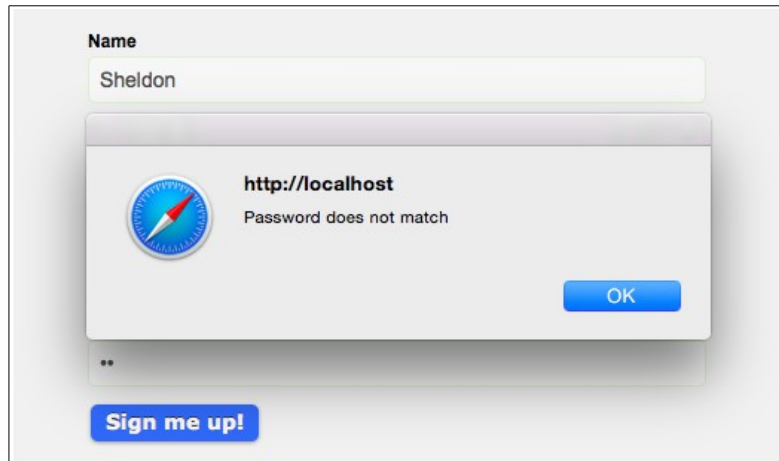
Sign me up!

Sign Up Page

Javascript function is used to validate the form data at client side. The following checks are performed:

1. Player has entered all the text fields and none of the fields are empty.
2. New password and confirm password fields match.
3. The email address is valid.

Validation is performed on clicking on sign up button. If player fails on any one of the criteria, an alert message with the problem is popped on the screen.



```
if((name == "") || (email == "") || (password == "") ||  
(repassword == "")) {  
    incomplete = true;  
    alert("All fields are required");  
}  
else if(!(password == repassword))  
{  
    incomplete = true;  
    alert("Password does not match");  
}  
  
else if (atpos ≤ 1 || ( dotpos - atpos < 2 ))  
{  
    incomplete = true;  
    alert("Invalid Email Id");  
}
```

If the information entered satisfies the above criteria AJAX is used to send asynchronous request to server to create new user information in database.

```
xmlhttp = new XMLHttpRequest();  
xmlhttp.open("GET", "../signUp.php?  
name="+name+"&email="+email+"&password="+password,true);  
xmlhttp.onreadystatechange=function() {  
    if (xmlhttp.readyState==4 && xmlhttp.status==200) {  
        if(xmlhttp.responseText.trim()=="success"){  
            alert("Successfully Signed Up!!");  
            window.location = "../HTML/tetrisGame.html";  
        }  
        else if(xmlhttp.responseText.trim()=="already")  
            alert("Username Already Exists");  
    }  
}  
xmlhttp.send(null);
```

On server side the client request for new user sign up is handled by PHP function which creates connection with database and sends SQL query to MYSQL server. It first checks if there is already a user with same email id. If yes, then server sends message to client informing the user with this email id already exist in system and player is prompted to use a different email id for registration. If the user does not exist then a new entry is inserted in to playerInfo table in database through PHP.

```
<?php

$con = mysql_connect('localhost', 'root' ,null) or die("Failed to connect
to MySQL: " . mysql_error());
$db = mysql_select_db('Tetris',$con) or die("Failed to connect to MySQL: " .
mysql_error());
if (!$con) {
    die('Could not connect: ' . mysqli_error($con));
}
$name = trim($_GET['name']);
$email = trim($_GET['email']);
$password = trim($_GET['password']);


$query1 = "SELECT * from playerInfo WHERE email='$email'";
$result1 = mysql_query($query1);
$count = mysql_num_rows($result1);

if($count == 0)
{
    $query2 = "INSERT INTO playerInfo VALUES('$email','$password','$name')";
    $result2 = mysql_query($query2);
    if($result2)
        echo "success";
    else
        echo "failed";
}
else
    echo "already";
mysql_close();

?>
```


Name

Email

**http://localhost**
Username Already Exists

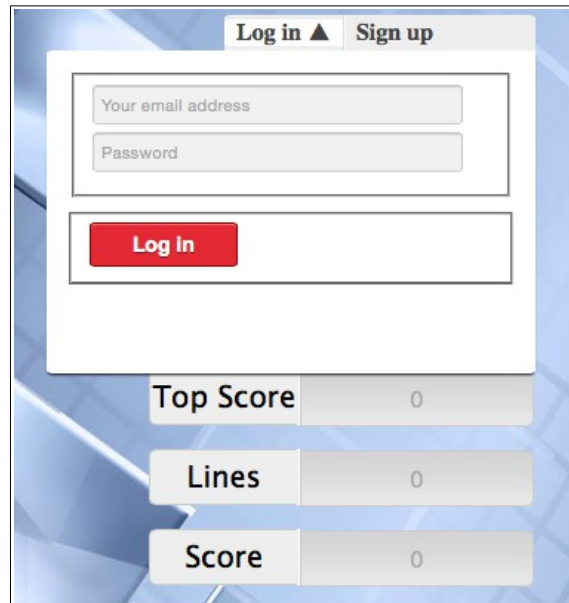
Name

Email

**http://localhost**
Successfully Signed Up!!

Player Login

An existing player can login to see previous scores, save new game scores and see top score. On clicking Log in a drop down form is shown asking player to enter email address and password. Validation is done in Javascript to check if any of the fields is empty or not. AJAX is used to generate request to server for login.



```
function checkLogin() {
    var u=document.getElementById("username").value;
    var p=document.getElementById("password").value;
    if(u == '' || p == '')
    {
        document.getElementById("errorStr").style.visibility = 'visible';
    }
    else{
        xmlhttp = new XMLHttpRequest();
        xmlhttp.open("GET","../connectivity.php?username="+u+"&password="+p,true);

        xmlhttp.onreadystatechange=function() {
            if (xmlhttp.readyState==4 && xmlhttp.status==200) {
                alert(xmlhttp.responseText.trim());

                var obj = jQuery.parseJSON(xmlhttp.responseText);

                if(obj[0].result == "success")
                {
                    document.getElementById("login").style.display='none';
                    document.getElementById("signup").style.display='none';

                    document.getElementById("inscore").style.visibility = 'visible';
                    document.getElementById("prescore").style.visibility = 'visible';
                    document.getElementById("signout").style.visibility = 'visible';
                }
            }
            else{
                document.getElementById("errorStr").style.visibility = 'visible';
            }
        }
    }
}
```

```

document.getElementById("level").value = obj[0].Score;
    }
    }
    xmlhttp.send(null);
    }
    if(isGameOver)
        saveScore();
}

```

On server side, PHP is used for login request. The PHP function starts a new user session, connects to database and sends query to retrieve rows with email and password provided by player. If the query returns one row count then the user login is successful and is conveyed back to client along with the top score among all the user. JSON encode-decode is used to aggregate success status and the top score.

```

<?php
session_start();
$con = mysql_connect('localhost', 'root' ,null) or die("Failed to connect to
MySQL: " . mysql_error());
$db = mysql_select_db('Tetris',$con) or die("Failed to connect to MySQL: " .
mysql_error());
if (!$con) {
    die('Could not connect: ' . mysqli_error($con));
}

$u = trim($_GET['username']);
$p = trim($_GET['password']);
$count = 0;
$query = sprintf("SELECT * FROM playerInfo where Email = '%s' AND Password =
'%s'",
                    mysql_real_escape_string($u),
                    mysql_real_escape_string($p)) ;

$result=mysql_query($query);

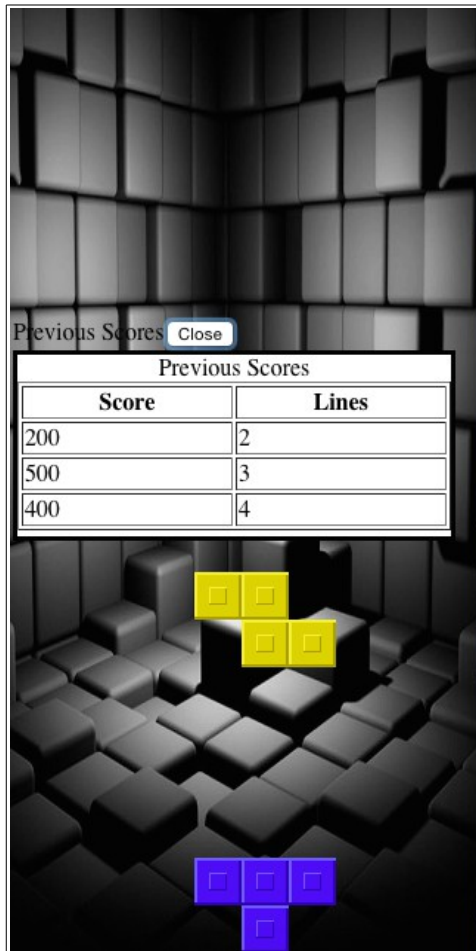
$count=mysql_num_rows($result);
$query1 = "SELECT max(Score) AS max_score FROM score";
$result1= mysql_query($query1);
$row = mysql_fetch_array($result1);
if($count==1){
    $_SESSION["username"] = $u;
    $output = "[";
    $output .= '{"result":"" . "success" . "',';
    $output .= '"Score":"" . $row["max_score"] . "'}';
    $output .= "]";
}
else
{
    $output = array('result'=>'failure',
                    'max_score'=>'0');
}

echo($output);
mysql_free_result($result);
mysql_close();
?>

```

Previous Scores

A player can retrieve all his/her previous scores by logging in and clicking the Previous Scores button. Again, a request is sent to server using AJAX and server retrieves all previous scores from database and returns back an array with all the previous scores. On client side javascript is used to dynamically generate a popup to display all previous scores.



```
function getScore() {  
  
    xmlhttp = new XMLHttpRequest();  
    xmlhttp.open("GET", "../getScore.php", true);  
  
    xmlhttp.onreadystatechange=function() {  
        if (xmlhttp.readyState==4 && xmlhttp.status==200) {  
            var obj = jQuery.parseJSON(xmlhttp.responseText);  
  
            var retStr = "";  
  
            retStr+="




```

```

for(i=0;i<obj.length;i=i+1)
{
    retStr+= "<tr><td>";
    retStr+= obj[i].Score;
    retStr+= "</td><td>";
    retStr+= obj[i].Lines;
    retStr+= "</td></tr>";
}
retStr+="  
</table>";
$(function() {
    $("#dialog").html(retStr);
    var popup = document.getElementById("dialog");
    popup.style.backgroundColor="white";
    popup.style.border="solid";
    $( "#dialog" ).dialog();
});
}
}
xmlhttp.send(null);
}

<?php
session_start();
$con = mysql_connect('localhost', 'root' ,null) or die("Failed to connect to MySQL: " .
mysql_error());
$db = mysql_select_db('Tetris',$con) or die("Failed to connect to MySQL: " .
mysql_error());
if (!$con) {
    die('Could not connect: ' . mysqli_error($con));
}
$user = $_SESSION["username"];
$query = "SELECT * FROM score WHERE Email='$user'";
$result = mysql_query($query);

$outp = "[";
while($row = mysql_fetch_array($result))
{
    if ($outp != "[") {$outp .= ",";}
    $outp .= '{"Score":"' . $row["Score"] . '","';
    $outp .= '"Lines":"' . $row["Lines"] . '"}';
}
$outp .= "]";
mysql_close();
echo($outp);

?>

```

```
<?php
session_start();
$con = mysql_connect('localhost', 'root' ,null) or die("Failed to connect to MySQL: " .
mysql_error());
$db = mysql_select_db('Tetris',$con) or die("Failed to connect to MySQL: " .
mysql_error());
if (!$con) {
    die('Could not connect: ' . mysqli_error($con));
}
$user = $_SESSION["username"];
$query = "SELECT * FROM score WHERE Email='$user'";
$result = mysql_query($query);

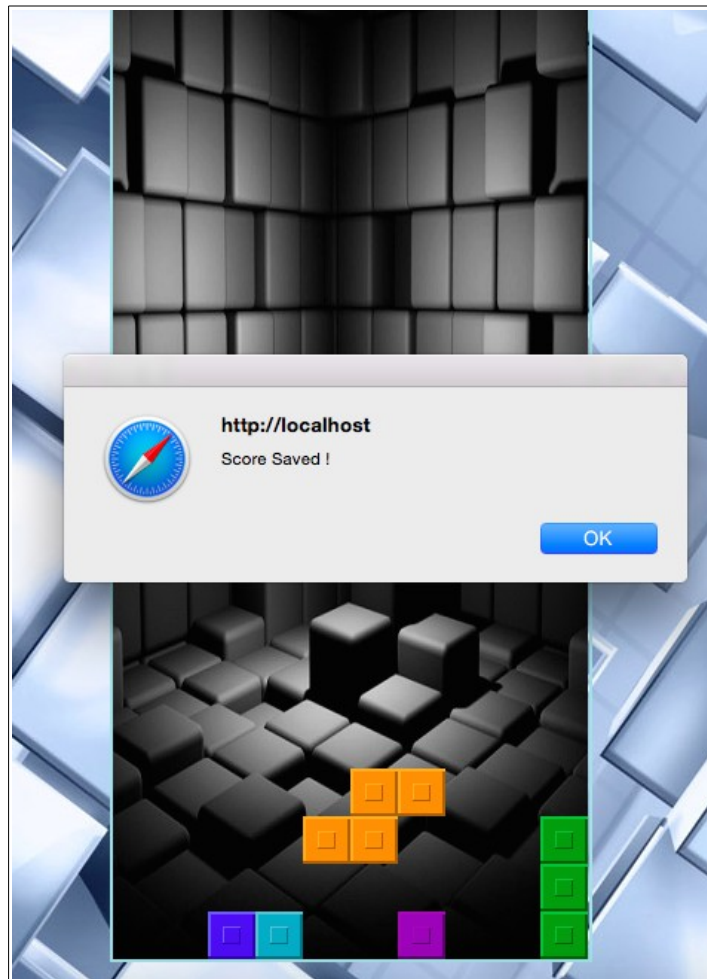
$outp = "[";
while($row = mysql_fetch_array($result))
{
    if ($outp != "[") {$outp .= ",";}
    $outp .= '{"Score":"' . $row["Score"] . '","';
    $outp .= '"Lines":"' . $row["Lines"] . '"}';
}
$outp .= "]";
mysql_close();
echo($outp);

?>
```

Save Score

Player can save his score at any given time by just clicking on Save Score button. A javascript function is invoked on button click which in turn use AJAX to send request to server along with username, score and lines information. On server side a PHP method connect to DB and

inserts the information in table along with current server time. If there is no error encountered then server notifies client with success message.



```
function saveScore(){
    var numLine = document.getElementById("lines").value;
    var score = document.getElementById("score").value;

    xmlhttp = new XMLHttpRequest();
    xmlhttp.open("GET", "../scoreUpdate.php?lines="+numLine+"&score="+score,true);

    xmlhttp.onreadystatechange=function() {
        if (xmlhttp.readyState==4 && xmlhttp.status==200) {
            if(xmlhttp.responseText.trim()=="success")
            {
                alert("Score Saved !");
            }
        }
    }
    xmlhttp.send(null);
}
```

```

<?php
session_start();
$con = mysql_connect('localhost', 'root' ,null) or die("Failed to connect to MySQL: " .
mysql_error());
$db = mysql_select_db('Tetris',$con) or die("Failed to connect to MySQL: " . mysql_error());
if (!$con) {
    die('Could not connect: ' . mysqli_error($con));
}
$lines = trim($_GET['lines']);
$score = trim($_GET['score']);
$user = $_SESSION["username"];

$query = "INSERT INTO score VALUES(DEFAULT,'$user','$score',now(),'$lines')";

$result=mysql_query($query);

if($result)
    echo "success";
else
    echo "failed";

mysql_close();

?>

```

Background Music

While the game is being played a background Tetris music is played. The music is inserted in HTML

```

<audio id="audio" loop>
  <source src="../images/Tetris.mp3">
  <source src="../images/Tetris.ogg">
</audio>

```

Help Page

A help page is developed for new players to get accustomed to game. Instructions can be read by clicking on Help button on main page.

Instructions for Tetris

[Go Back](#)

The Matrix

Tetris is played on a 10 by 20 grid called theMatrix. Shapes called Tetriminos fall from the top of the Matrix and come to rest at the bottom. Only one Tetrimino falls at a time. At first the Tetriminos fall rather slowly; as the game progresses they will fall faster and faster.

Moving the Tetriminos

Using the arrow keys, you can adjust where and how the Tetriminos fall. By pressing the LEFT and RIGHT Arrow keys, you can slide the falling Tetrimino from side to side. You can't slide a Tetrimino past the edge of the Matrix. By pressing the UP Arrow key, you can rotate the Tetrimino 90 degrees clockwise. You can move the Tetriminos even after they land at the bottom of the Matrix briefly. The Tetrimino will Lock Down as soon as you stop trying to move it. At that point, the next Tetrimino will begin to fall.

Hard Drop and Soft Drop

As your game improves and you begin to pay attention to the Tetriminos in your Next Queue, you might find yourself getting a little impatient waiting for the piece to fall. You have two options to speed up the game—the Soft Drop and the Hard Drop—and they're both really easy to do.

The Soft Drop is performed by pressing the DOWN Arrow key—the Tetrimino will fall much faster than usual while you hold down the key, but as soon as you let go the piece will resume its normal pace. You retain complete control over the piece while doing a Soft Drop.

The Hard Drop is much less forgiving—hit the Space Bar to cause the Tetrimino to fall straight down, forgoing any further opportunity to move it. The Hard Drop is great for timed games where your goal is to get pieces into position as quickly as possible. Pay attention to the Ghost Piece to help you see where the Tetrimino will fall, and don't press the space bar until you're ready!

Database

The database schema of Tetris is composed of two tables, one containing Score information and other containing Player information. The Player information table (playerInfo) consists of fields Name, Email-Id and Password where Email-Id is taken as Primary key and Name and Password is set to constraint Not Null. The table score comprises of fields idscore, Score, Lines, Email and Time. The column idscore is taken as primary key which is auto generated for each new score saved by the player. The column Email serves as foreign key to the field Email of table playerInfo thus linking the player information with his score information. The time at which a player saves the Tetris game score is recorded in the Time field. The number of lines and total score made by the player is recorded in Lines and Score fields respectively.

Result Grid			
Filter Rows:		Q Search	Edit: Export/Import
Email	Password	Name	
saurabh.mathur@outlook.com	eternal	Saurabh Mathur	
scooper@gmail.com	shelly	Sheldon Cooper	
ritz@gmail.com	ritrocks	Ritika Singh	
gregory@yahoo.com	housegreg	Gregory Cuddy	
NULL	NULL	NULL	

Table playerInfo

Result Grid					
Filter Rows:		Q Search	Edit:	Export/Import:	
idscore	Email	Score	Time	Lines	
32	scooper@gmail.com	100	2014-12-10 0...	1	
33	a	200	2014-12-10 0...	2	
34	shelly@yahoo.com	200	2014-12-11 0...	2	
35	gregory@yahoo.com	800	2014-12-11 0...	3	
36	ritz@gmail.com	300	2014-12-11 0...	3	
37	saurabh.mathur@outlook...	1200	2014-12-12 0...	4	
NULL	NULL	NULL	NULL	NULL	

Table Score