Pits Of Doom Lesson 7: Join, Login & Lost Password

In the last lesson we created a database class to make doing queries quick and easy to handle. Then we converted over the current character file and map editor to use the database when storing/retrieving information. Finally, I added an additional file so you could import your map.txt files into your database from any you made in the previous lessons.

Now it's time to make our game start to feel like a real game. In this lesson we'll create two new classes, a member's class and a character's class. Then we'll create two new tables, a member's table and a character's table. Then we'll create a join, login and lost password page so members can join, login, and access their own personal account and personal character. Last but not least, we'll load characters that belong only to the member whose logged in and show those in the game map.

Lesson Concepts

Database Tables
In order to figure out if someone is a member of our game or not we have to store all the information in the database. This way we can pull it out any time someone tries to play the game using PHP. Just like we did with the map tables, we need to plan out the member and character tables. Think of the essential things we need to know about members and characters then draft it out like this:

Members
ID (number) – a unique number assigned to all members

Username (string) – must be at least 3 characters long, no longer then 30
cannot be changed by member

Password (string) – must be at least 5 characters long, no longer then 15
can be changed by member

Use both their username and password to login to the game

Email (string) – so we can send them their account info if they can't remember it

Last Login (date & time) – we can keep track of the last time they logged in

Online (true or false) – we set this once they login and unset it once they logout

Money (number) – how much money they have in the game, start off with 500G

Current Character (number id of the character) – the character they're playing right now

Banned (true or false) – if they're banned we can make sure they can't login and play

If we think of anything else we can add it later. If you can think of something you want on your game that is specifically for members go ahead and add it here now.

Characters
ID (number) – each character has a unique number

Member ID (number) – the ID of the member this character belongs to

First Name – must be at least 3 letters long, no longer then 15
cannot be changed after being set by member

Last Name – must be at least 5 letters long, no longer then 15
cannot be changed after being set by member

*NO 2 CHARACTERS CAN HAVE THE SAME FIRST & LAST NAME!!

Type (id of a type) – human, dragon, elf, ogre, dwarf, vampire, mage, whatever, we want to add lots of these so it'd be best to make another table and pull in the ID of the type from there.

Monster (true or false) – if this is set to true, the game automatically plays this character otherwise the character will be played by whoever it belongs to (member id).

Money (number) – characters will pickup money as they play

Health (number) – 0 to 100, when a character's health is 0 they're dead

Speed (number) – 0 to 100 how fast the character moves

Intelligence (number) – 0 to 100 how smart the character is

Strength (number) – 0 to 100 how powerful the character's blows are

Agility (number) – 0 to 100 how evasive the character is

Magic (number) – 0 to 100 strength of their magic

Fighting Level (number) – starts at 0, increases after every character they kill above or equal to their fighting level

Magic Level (number) – starts at 0, increases every time they defeat someone using magic of the same or higher magic level

Magic Amount (number) – starts at 100, how much magic they have they can currently used

Weapon Equipped (id of a item) – the weapon in they're using

Spell Equipped (id of a item) – the spell they're using

Armor Equipped (id of item) – the armor they're wearing

Shield Equipped (id of a item) – the shield they're using

Map (id of a map) – the map they're currently on

X (number) – their x position on the current map

Y (number) – their y position on the current map

Z (number) – their level on the current map

Active (true or false) – this automatically turns on when the player logs into the game and off when they logout and the character isn't fighting

Attacking (number) – ID of the character or monster they're attacking

Character Types
Sometimes you'll find as you make a game adding an additional table is a HUGE help. In this case, we can easily add as many different character types to the game as we want and set the default range values they should have when they're created. By giving each random statistics as soon as they're made, no two will be the same.

ID (number) – a unique ID of the type of character

Name (string) – the name of this character type

To make things easier for monster creation, we can setup the minimum/maximum values each of these types should have when they're created in the game. For instance, an Elf type would be faster then a Dwarf type.

MinHealth (number)

MaxHealth (number)

MinSpeed (number)

MaxSpeed (number)

MinIntelligence (number)

MaxIntelligence (number)

MinStrength (number)

MaxStrength (number)

MinAgility (number)

MaxAgility (number)

MinMagic (number)

MaxMagic (number)

Now that we have our three tables sketched out we can go ahead and generate the SQL for them. That would look like this:

```
mysql > CREATE TABLE members (
    id integer NOT NULL AUTO_INCREMENT,
    username varchar(30) NOT NULL,
    password varchar(15) NOT NULL,
    email varchar(50) NOT NULL,
    lastlogin datetime,
    online bool,
    money integer DEFAULT 500,
    curcharacter integer,
    banned bool,
    UNIQUE KEY(id)
 );

mysql > CREATE TABLE characters (
    id integer NOT NULL AUTO_INCREMENT,
    memberid integer NOT NULL,
    firstname varchar(15) NOT NULL,
    lastname varchar(15) NOT NULL,
    type integer NOT NULL,
    monster bool NOT NULL,
    money integer NOT NULL,
    health integer NOT NULL,
    speed integer NOT NULL,
    intelligence integer NOT NULL,
    strength integer NOT NULL,
    agility integer NOT NULL,
    magic integer NOT NULL,
    fightlvl integer NOT NULL,
    magiclvl integer NOT NULL,
    magicamount integer NOT NULL,
    weapon integer NOT NULL,
    spell integer NOT NULL,
    armor integer NOT NULL,
    shield integer NOT NULL,
    mapid integer NOT NULL,
    x integer NOT NULL,
    y integer NOT NULL,
    z integer NOT NULL,
    active bool NOT NULL,
    attacking integer NOT NULL,
    UNIQUE KEY(id)
 );

mysql > CREATE TABLE character_types (
    id integer NOT NULL AUTO_INCREMENT,
    name varchar(50) NOT NULL,
    minhealth integer NOT NULL,
    maxhealth integer NOT NULL,
    minspeed integer NOT NULL,
    maxspeed integer NOT NULL,
    minintelligence integer NOT NULL,
    maxintelligence integer NOT NULL,
    minstrength integer NOT NULL,
    maxstrength integer NOT NULL,
    minagility integer NOT NULL,
    maxagility integer NOT NULL,
    minmagic integer NOT NULL,
    maxmagic integer NOT NULL,
    UNIQUE KEY(id)
 );
```

Creating The Classes
Now that we have our tables I find it easiest to create classes for them. This way, once we start programming, we don't even have to worry about any of the database stuff that's going on in the back end. I find it easier to see the big picture and put the smaller things out of mind as much as possible. So that's what we'll do now. In order to use these classes, we must have already established a connection to the database.

For the members class, the easiest thing to do is think of all the things you DO as a member. You join, login, change your password, pick a character to play as, and so forth. These are all things we'll need to incorporate into our members class. For right now we'll only do implement the things we need to join, login, and send for a lost password but you'll see a lot more of this class as the game progresses.

I have several rules of thumb for my classes. I've found following these basic tips takes a lot of stress out of making a game:

1) Always comment!! If you look at your code later and don't understand what's going on, or why you did what you did, sometimes it takes hours — or days even — to retrace your steps. Give yourself clear cut comments on what a function does and why it does it. No, you don't have to write a book but you should have enough information there so you can come back later and figure it out in under 5 minutes.

2) Name your class variables the same as your database field names. This may seem like a "duh" kind of comment but there are two reasons this is helpful. First, you'll know what the field name is when you write queries in a class because all the fields are right at the top of the page. Second, we can do some really nifty database queries and assign the values right back into the variables because they both have the same name.

3) When in doubt think it out. The reason we're making classes for our game is to help separate things out and join things together (a contradiction, I know!!). Don't make a class do something do something that seems awkward. For example, the member doesn't fight a monster, the character does. A character doesn't have pits or walls, the map does. It wouldn't make sense to give the member class a fight() function or the character class a hasPit(x, y, z) kind of function.

Member Class

```
<?php
/**************
* File: memberobj.php
* Purpose: member class
**************/

class member
{

    //these are named exactly the same as in the database
    //to make it easier when we use the $database class later
    var $id;
    var $username;
    var $password;
    var $email;
    var $lastlogin;
    var $online;
    var $money;
    var $curcharacter;
    var $banned;
    var $database;
```

```
//these values are for our database queries
var $table;
var $where;

//these are arrays we keep for this member
var $characters = array(); //a list of all characters they own

/**************
* Default constructor, load this member's information
* Remember, this always has to match the name of the class
**************/
function member($id)
{
 //we need to get access to the database class
 global $_SESSION;

   if (!$id || !isNumeric($id)) //don't try to do any of this if there's no id number
      return null;

   $this->id = $id;
   $this->database = $_SESSION['database'];
   $this->table = "members"; //the name of the table we're using in our database when we
                          //run any queries on members
   $this->where = "WHERE id='$this->id'"; //we always want records from members with this ID

   //now we use the database class we made to pull in all the information about this member
   $this->username = $this->database->select("username", $this->table, $this->where);
   $this->password = $this->database->select("password", $this->table, $this->where);
   $this->email = $this->database->select("email", $this->table, $this->where);
   $this->lastlogin = $this->database->select("lastlogin", $this->table, $this->where);
   $this->online = $this->database->select("online", $this->table, $this->where);
   $this->money = $this->database->select("money", $this->table, $this->where);
   $this->curcharacter = $this->database->select("curcharacter", $this->table, $this->where);
   $this->banned = $this->database->select("banned", $this->table, $this->where);

  //last but not least, we have to load any characters they have
  //this uses a function in the database class I never talked about before
  //it goes through the database and finds the ID number of every character they own
  //then it creates a character object for each of those characters they own and puts
  //them into an array. Finally, it returns that array to the character class so we can
  //view all of the players characters.
  $this->characters = $this->database->loadArray("id", "characters", "WHERE memberid='$this->id'", "character");

  //!!!!! To really get an understanding of what this does, I suggest you uncomment the line
  //below and see what happens. It will print out all the information on each character the
  //member owns.

  //print_r($this->characters);

 } //end default constructor

 /*************
 * The member has supplied the correct login information, log them into the game
 *************/
 function login()
 {

   //remember to update the object value
   $this->lastlogin = time();

   //and the database value for the field
   $this->database->update($this->table, "lastlogin", $this->lastlogin);

   $this->online = true;
   $this->database->update($this->table, "online", $this->online);

   //make all their character's active now that they're online
   $this->database->update("characters", "active=true", "WHERE memberid='$this->id'");
 }

 /*************
 * The member is leaving the game
 *************/
 function logout()
 {

   $this->online = false;
   $this->database->update($this->table, "online", $this->online);

   //make all their character's active now that they're online
   $this->database->update("characters", "active=false", "WHERE memberid='$this->id'");
 }
} //end the members class
```

```
?>
```

Character Class

```
<?php
/**************
* File: characterobj.php
* Purpose: character class
**************/

class character
{

    //my rule of thumb is, always have variables
    //for every field in your database so you can access them
    //quickly and easily
    var $id;
    var $memberid;
    var $firstname;
    var $lastname;
    var $type;
    var $monster;
    var $money;
    var $health;
    var $speed;
    var $intelligence;
    var $strength;
    var $agility;
    var $magic;
    var $fightlvl;
    var $magiclvl;
    var $magicamount;
    var $weapon;
    var $spell;
    var $armor;
    var $shield;
    var $mapid;
    var $x;
    var $y;
    var $z;
    var $active;
    var $attacking;
    var $database;
```

```
//these values are for our database queries
var $table;
var $where;

/**************
* Default constructor, load this characters's information
*************/
function character($id)
{
 //we need to get access to the database class
 global $_SESSION;

   if (!$id || !isNumeric($id)) //don't try to do any of this if there's no id number
      return null;

   $this->id = $id;
   $this->database = $_SESSION['database'];
   $this->table = "characters";
   $this->where = "WHERE id='$this->id'";

   //now we use the database class we made to pull in all the information about this member
   $this->id = $this->database->select("id", $this->table, $this->where);
   $this->firstname = $this->database->select("firstname", $this->table, $this->where);
   $this->lastname = $this->database->select("lastname", $this->table, $this->where);
   $this->type = $this->database->select("type", $this->table, $this->where);
   $this->monster = $this->database->select("monster", $this->table, $this->where);
   $this->money = $this->database->select("money", $this->table, $this->where);
   $this->health = $this->database->select("health", $this->table, $this->where);
   $this->speed = $this->database->select("speed", $this->table, $this->where);
   $this->intelligence = $this->database->select("intelligence", $this->table, $this->where);
   $this->strength = $this->database->select("strength", $this->table, $this->where);
   $this->agility = $this->database->select("agility", $this->table, $this->where);
   $this->fightlvl = $this->database->select("fightlvl", $this->table, $this->where);
   $this->magiclvl = $this->database->select("magiclvl", $this->table, $this->where);
   $this->magicamount = $this->database->select("magicamount", $this->table, $this->where);
   $this->weapon = $this->database->select("weapon", $this->table, $this->where);
   $this->spell = $this->database->select("spell", $this->table, $this->where);
   $this->armor = $this->database->select("armor", $this->table, $this->where);
   $this->shield = $this->database->select("shield", $this->table, $this->where);
   $this->mapid = $this->database->select("mapid", $this->table, $this->where);
```

```
    $this->x = $this->database->select("x", $this->table, $this->where);
    $this->y = $this->database->select("y", $this->table, $this->where);
    $this->z = $this->database->select("z", $this->table, $this->where);
    $this->active = $this->database->select("active", $this->table, $this->where);
    $this->attacking = $this->database->select("attacking", $this->table, $this->where);

 } //end default constructor

 /**************
 * Move around the map (does this look familiar?!?)
 **************/
 function move($direction)
 {

     //we need to get the character's current location
$newx = $this->x;
$newy = $this->y;

switch($direction) //we want to change what we're checking
      //depending on the direction the character is moving
{
  case "right": $newy++; //add one to the y value
    break;
  case "left": $newy--; //subtract one from the y value
    break;
  case "back": $newx++; //add one to x value
    break;
  case "forward": $newx--; //subtract one from the x value
    break;
}

if (getValue($newx, $newy, $this->mapid, $this->z) == "L")
{
  //they are currently ON a ladder position
  //if they hit the up direction, move them up a level (if not at level 1)
  if ($direction == "forward" && $this->z != 1)
  {
    echo "You moved up the ladder!";

    //move them up a level
    $this->z = $this->z - 1;
  //set the character's starting position in the map
  $this->x = startPositionX($this->mapid, $this->z);
  $this->y = startPositionY($this->mapid, $this->z);

  //now we save their position to the database so if they log off
  //or leave the game the character is still in this position when
  //they come back and play later
  $this->database->update("character", "x=$this->x, y=$this->y, z=$this->z", $this->where);

}
else if ($direction != "back")
{
   //let them move some other direction
   if (getValue($newx, $newy, $this->mapid, $this->z) == "T")
   {
     //the treasure is in this direction
     echo "You found the treasure!";
     $this->x = $newx;
     $this->y = $newy;

     //update their position
     $this->database->update("character", "x=$this->x, y=$this->y", $this->where);
   }

   else if (getValue($newx, $newy, $this->mapid, $this->z) == "W")
   {
     //don't update their position, they can't move here
     echo "You hit a wall!";
   }

   else if (getValue($newx, $newy, $this->mapid, $this->level) == "E")
   {
     //empty space, move them to this new location
     echo "You moved $direction one space.";

     $this->x = $newx;
     $this->y = $newy;

     //update their position
     $this->database->update("character", "x=$this->x, y=$this->y", $this->where);
   }
```

```
    else if (getValue($newx, $newy, $this->mapid, $this->level) == "S")
    {
     //starting location, move them to this new location
     echo "You moved $direction one space.";

     $this->x = $newx;
     $this->y = $newy;

     //update their position
     $this->database->update("character", "x=$this->x, y=$this->y", $this->where);
    }

    else if (getValue($newx, $newy, $this->mapid, $this->level) == "X")
    {
     //they found a pit
     echo "You fell into a pit and dropped down a level!";

     //move them down a level
     $this->z = $this->z + 1;

     $this->x = startPositionX($this->mapid, $this->z);
     $this->y = startPositionY($this->mapid, $this->z);

     //update their position
     $this->database->update("character", "x=$this->x, y=$this->y, z=$this->z", $this->where);
    }

    else if (getValue($newx, $newy, $this->mapid, $this->z) == "L")
    {
     //they found a ladder
     echo "You found a ladder. Move up or down?";

     //move them to the position on the map that has the ladder
     //but don't change which level they're on
     $this->x = $newx;
     $this->y = $newy;

     //update their position
     $this->database->update("character", "x=$this->x, y=$this->y", $this->where);
    }
  }
```

```
//if they hit the down direction, move them down a level (if not at level 5)
if ($direction == "back" && $this->z != 5)
{
 echo "You moved down the ladder!";

 //move them down a level
 $this->z = $this->z + 1;

 //set the character's starting position in the NEW map
 $this->x = startPositionX($this->mapid, $this->z);
 $this->y = startPositionY($this->mapid, $this->z);

 //update their position
 $this->database->update("character", "x=$this->x, y=$this->y, z=$this->z", $this->where);
}
else if ($direction != "forward")
{
 //let them move some other direction
   if (getValue($newx, $newy, $this->mapid, $this->z) == "T")
   {
    //the treasure is in this direction
    echo "You found the treasure!";

    $this->x = $newx;
    $this->y = $newy;

    //update their position
    $this->database->update("character", "x=$this->x, y=$this->y", $this->where);

   }

   else if (getValue($newx, $newy, $this->mapid, $this->level) == "W")
   {
    //don't update their position, they can't move here
    echo "You hit a wall!";
   }

   else if (getValue($newx, $newy, $this->mapid, $this->level) == "E")
   {
    //empty space, move them to this new location
    echo "You moved $direction one space.";
    $this->x = $newx;
```

```
  $this->y = $newy;

  //update their position
  $this->database->update("character", "x=$this->x, y=$this->y", $this->where);
}

else if (getValue($newx, $newy, $this->mapid, $this->z) == "S")
{
  //starting location, move them to this new location
  echo "You moved $direction one space.";
  $this->x = $newx;
  $this->y = $newy;

  //update their position
  $this->database->update("character", "x=$this->x, y=$this->y", $this->where);
}

else if (getValue($newx, $newy, $this->mapid, $this->z) == "X")
{
  //they found a pit
  echo "You fell into a pit and dropped down a level!";

  //move them down a level
  $this->z = $this->z + 1;

  $this->x = startPositionX($this->mapid, $this->z);
  $this->y = startPositionY($this->mapid, $this->z);

  //update their position
  $this->database->update("character", "x=$this->x, y=$this->y, z=$this->z", $this->where);
}

else if (getValue($newx, $newy, $this->mapid, $this->z) == "L")
{
  //they found a ladder
  echo "You found a ladder. Move up or down?";

  //move them to the position on the map that has the ladder
  //but don't change which level they're on
  $this->x = $newx;
  $this->y = $newy;
```

```
        //update their position
        $this->database->update("character", "x=$this->x, y=$this->y", $this->where);
      }
  }
}

else if (getValue($newx, $newy, $this->mapid, $this->z) == "T")
{
  //the treasure is in this direction
  echo "You found the treasure!";

  $this->x = $newx;
  $this->y = $newy;

  //update their position
  $this->database->update("character", "x=$this->x, y=$this->y", $this->where);
}

else if (getValue($newx, $newy, $this->mapid, $this->z) == "W")
{
  //don't update their position, they can't move here
  echo "You hit a wall!";
}

else if (getValue($newx, $newy, $this->mapid, $this->z)== "E")
{
  //empty space, move them to this new location
  echo "You moved $direction one space.";

  $this->x = $newx;
  $this->y = $newy;

  //update their position
  $this->database->update("character", "x=$this->x, y=$this->y", $this->where);
}

else if (getValue($newx, $newy, $this->mapid, $this->z) == "S")
{
  //starting location, move them to this new location
  echo "You moved $direction one space.";

  $this->x = $newx;
```

```
    $this->y = $newy;

    //update their position
    $this->database->update("character", "x=$this->x, y=$this->y", $this->where);
  }

  else if (getValue($newx, $newy, $this->mapid, $this->z) == "X")
  {
    //they found a pit
    echo "You fell into a pit and dropped down a level!";

    //move them down a level
    $this->z = $this->z + 1;

    //set the character's starting position in the map
    $this->x = startPositionX($this->mapid, $this->z);
    $this->y = startPositionY($this->mapid, $this->z);

    //update their position
    $this->database->update("character", "x=$this->x, y=$this->y, z=$this->z", $this->where);
  }

  else if (getValue($newx, $newy, $this->mapid, $this->z) == "L")
  {
    //they found a ladder
    echo "You found a ladder. Move up or down?";

    //move them to the position on the map that has the ladder
    //but don't change which level they're on
    $this->x = $newx;
    $this->y = $newy;

    //update their position
    $this->database->update("character", "x=$this->x, y=$this->y", $this->where);
  }

} //end the character class
?>
```

Character Types Class

```
<?php
/*************
* File: charactertypeobj.php
* Purpose: character types
*************/

class charactertype
{

    var $id;
    var $name;
    var $minhealth;
    var $maxhealth;
    var $minspeed;
    var $maxspeed;
    var $minintelligence;
    var $maxintelligence;
    var $minstrength;
    var $maxstrength;
    var $minagility;
    var $maxagility;
    var $minmagic;
    var $maxmagic;
    var $database;

   //these values are for our database queries
   var $table;
   var $where;

   /*************
   * Default constructor, load character type
   *************/
   function charactertype($id)
   {
    //we need to get access to the database class
    global $_SESSION;

      if (!$id || !isNumeric($id)) //don't try to do any of this if there's no id number
         return null;

      $this->id = $id;
      $this->database = $_SESSION['database'];
      $this->table = "character_types"; //the name of the table we're using in our database when we
                      //run any queries on members
      $this->where = "WHERE id='$this->id'"; //we always want records from members with this ID

      //now we use the database class we made to pull in all the information about this member
      $this->name = $this->database->select("name", $this->table, $this->where);
      $this->minhealth = $this->database->select("minhealth", $this->table, $this->where);
      $this->maxhealth = $this->database->select("maxhealth", $this->table, $this->where);
      $this->minspeed = $this->database->select("minspeed", $this->table, $this->where);
      $this->maxspeed = $this->database->select("maxspeed", $this->table, $this->where);
      $this->minintelligence = $this->database->select("minintelligence", $this->table, $this->where);
      $this->maxintelligence = $this->database->select("maxintelligence", $this->table, $this->where);
      $this->minstrength = $this->database->select("minstrength", $this->table, $this->where);
      $this->maxstrength = $this->database->select("maxstrength", $this->table, $this->where);
      $this->minagility = $this->database->select("minagility", $this->table, $this->where);
      $this->maxagility = $this->database->select("maxagility", $this->table, $this->where);
      $this->minmagic = $this->database->select("minmagic", $this->table, $this->where);
      $this->maxmagic = $this->database->select("maxmagic", $this->table, $this->where);

   } //end default constructor

} //end the character type class
?>
```

Join Script
We don't want just anyone to be able to access characters and maps in the game, they should only see things that belong to them. In order to do this we'll have them make an account in our database and automatically create a character for them to start playing with. When we do this we'll generate random statistics for the character based on the data we have in our character types data.

Lost Password Script
It's inevitable, eventually you'll forget the information you entered when you joined the site. Creating a lost password script will remedy this situation. Now your members can have their login & password sent directly to their email address instead of bogging down your inbox with pointless password requests!

Login Script
Time to verify, are you who you say you are? The biggest downfall of a login is that anyone who has the correct login/password combination will be able to access the information for the account, but that's something we've come to accept as unavoidable with online account security. However, we can make things as secure as possible by only allowing members who have logged in to view certain parts of the website. We'll enhance that even further by only giving them access to records they own or created themselves.

```
$database = & new database($host, $username, $password, $db);

if (isset($_POST['submit'])) {
	//check to see if this login is in the members table
	//by retrieving their member id if their username and password matches
	$id = $database->single("id", "members", "WHERE username='" . $_POST['username'] . "' AND
password='" . $_POST['password'] . "'");

	//we found the member, let's create a session with this member's information
	if ($id)
	{
		$_SESSION['database'] = $database;
		$_SESSION['member'] = new member($id);
		//log them into the game
		$_SESSION['member']->login();

		//now let's redirect them to the map and their character
		header("Location: character_v6.php");
		exit;
	}
	//we didn't find the login, we won't let them login
	$error = errorMsg("Incorrect login information");
 }
```

The logic behind this script is fairly straightforward. First we connect to the database, then if they've hit the submit button we pass the username and password they entered into the form to our member object. If we find a member ID that matches that information we give them access to the game by storing the member object into a session. Otherwise their login information was incorrect and they need to try again.

Summary

Try the working version! In this lesson we added a huge backbone to what is going to quickly become a working, running game. Now we've restricted access to the map view to people who have logged in. In order to be logged in you must have an account in our database — thus we created a join script people can use to create a default character and setup their own unique login and password. Finally, to top it off, we made a lost password form so anyone who forgets their login information can have it emailed to them.

Game Files

Working Version:

Pits of Doom, use the login below or sign up and create your own account!

Login: test

Password: test

Source Files:
classfiles.php
functions.php

Object Files:
mysqlobj.php
memberobj.php
characterobj.php
charactertypeobj.php

Game Files
login.php
join.php
lostpassword.php
character.php

SQL Data:
pitsofdoom.sql — includes character types and default map

Lesson Reasoning

This lessons is all about planning out the underlying structure of members and characters and how they interact with the game map. Each member can have multiple characters. In order to do that we have to prompt the user for their login information: this way we can verify they are who they say they are. Once we know who the member is, we load all the information from the database about the characters they have. Finally, we display their character's information in the map depending on the information we have about the character in our database.

In the next lesson we'll create a way for members to create their own characters, instead of automatically creating one for them when they join. We'll also start working on monsters (!!!!) and letting character's battle them in the map.

Find lesson 8 at http://design1online.com!