

Torque Game Builder – Checkers Tutorial - Part 4

4. Scripting our Game Start

4.1 Creating our base CheckerBoard object

Before we start our game process we will create a base CheckerBoard object with some functions that will be needed. In our game we will have a ServerCheckerBoard that simply holds numerical data that represents the locations of pieces. We will also have two ClientCheckerBoards, one on each client, that will store the same numerical data as well as the visual image data. We will have a base class and function set that both the ServerCheckerBoard class and the ClientCheckerBoard class will inherit from (we can simulate inheritance in TGB through ScriptClasses). The base CheckerBoard class will have functions for checking if a move is legal. This way the client can first check if the move is legal based on its simulated ClientCheckerBoard. If so, then it will send a request for that move to the server, then the server will check if the move is legal on its ServerCheckerBoard and pass back whether the move was done. This way we can attach the function to check if a move is legal on the CheckerBoard base class and both the ClientCheckerBoard and the ServerCheckerBoard will have access to that same function. This allows the client to not request moves it already knows are invalid and will cut down on network traffic while still enabling the server to be the authoritative source.

We are going to create a *ScriptClass* called CheckerBoard and then add functions attached to the CheckerBoard class (put this in your checkerboard.cs file).

```
new ScriptClass(CheckerBoard);

function CheckerBoard::isBlack(%this, %x, %y)
{
    // this will decide if a checkerboard slot is black
    if( (%x % 2) && (%y % 2) )
        return true;
    else if( %x % 2 || %y % 2 )
        return false;
    else
        return true;
}
```

Code Sample 4.1.1

This function (as is obvious by the name) will determine if a checkerboard slot is black. It uses basic math comparisons to see if that checkerboard position is divisible by 2 or not, to invert the starting checkerboard slot you would switch all the true statements with false and vice-versa.

Now we begin with our data setting and accessing functions by adding set, get, and remove piece (in your checkerboard.cs file).

```
function CheckerBoard::setPiece(%this, %x, %y, %type)
{
    // set the board's array position
    %this.board[%x,%y] = %type;
}
```

Torque Game Builder – Checkers Tutorial - Part 4

```
function CheckerBoard::getPiece(%this, %x, %y)
{
    // return the piece stored at the position passed
    return %this.board[%x,%y];
}

function CheckerBoard::removePiece(%this, %x, %y)
{
    // set the board position to none
    %this.board[%x,%y] = $none;
}
```

Code Sample 4.1.2

We don't do anything too special here, we just access a “board” array attached to our CheckerBoard (represented). This array has an x and y component to represent the x and y axis of our checkerboard. The setPiece function is passed an x position, y position, and type. It then sets the array position passed in as x and y to the type passed in. The type will be either 0, 1, 2, 3, or 4 represented by \$none, \$red, \$blue, \$redKing, or \$blueKing. This x and y position will be simple integers, not the actual image position, just the logical position of the piece on the checkerboard. The getPiece function just returns the type stored in the array location of the x and y passed in. The removePiece function just sets the array position of the x and y passed in to 0 or \$none.

Now we get to the meat of our game play. We are going to add the isLegalMove() function which will determine if the a move from the first x and y positions (fromX and fromY) to the second set of x and y positions (toX and toY) is valid. This function is fairly large and a bit scary at first so we will break it down into chunks and go over each part. This is the first chunk.

```
function CheckerBoard::isLegalMove(%this, %fromX, %fromY, %toX, %toY)
{
    // check if the move is to the same location,
    // if so we simply return "same"
    if(%fromX == %toX && %fromY == %toY)
    {
        return "same";
    }
}
```

First we create our function name and the values to pass in. One thing you may or may not know is that when you attach a function to an object's namespace with “::” the first value passed in is always %this, which represents the object the method is called on.

Here we first check if the fromX and fromY is the same as toX and toY, if so we return “same” since the checker piece is trying to move to the same piece it just came from. Here is our next chunk.

```
// What color is this piece anyhow?
%piece = %this.getPiece(%fromX, %fromY);

// The trivial case, make sure the square is black.
if(!%this.isBlack( %toX, %toY))
    return false;
```

Torque Game Builder – Checkers Tutorial - Part 4

```
// Now lets do another trivial check, lets make sure there
// isn't a piece already in the spot we're trying to move to
%moveToSlot = %this.getPiece(%toX, %toY);

// If there is a piece there we can't move there
if(%moveToSlot != $none)
    return false;
```

The first thing we do in this chunk is grab the piece with our recently created `getPiece()` function. We then check if the slot we are trying to move to is a black slot or not, as we know with checkers rules a piece cannot move to a non-black piece. We next get the slot we are trying to move to. This is the function that we just created and will give us the type that's in that slot, including if that slot is empty. We then check to make sure that the slot is equal to `$none`. If not we exit out. On to our next chunk.

```
// Check to make sure the move is in the right direction
switch(%piece)
{
    case 0: // Sanity check
        return false;

    case 1: // Red team, going in the positive Y direction
        if(%fromY > %toY)
            return false;
    case 2: // Blue team, going in negative Y direction
        if(%fromY < %toY)
            return false;

    // If it's a king, it will skip this, because kings can move in any
    // direction
}
```

This chunk of code will do some checks to make sure we are moving in the right direction. First, if the piece is none we exit out. If the piece is red, then it makes sure the `toY` is greater than the `fromY`. This makes sure the piece is headed downward. If the piece is blue, we make sure its `toY` is less than the `fromY` so we are only heading upward. This means if the piece is not none, red, or blue then it can basically move in any direction, which is what we want since the only other pieces are the `redKing` and `blueKing`. Add this as our next chunk of code.

```
// Now the tougher one, make sure the distance traveled is legal
%xDistance = mAbs(%fromX - %toX);
%yDistance = mAbs(%fromY - %toY);

// Trivial case, the distance traveled is one.
if(%xDistance == 1 && %yDistance == 1)
{
    return true;
```

First we grab the absolute distance that our from and to locations have. We then check if both the x and y distances are 1. If so we return true. This means that this piece is simply moving one

Torque Game Builder – Checkers Tutorial - Part 4

unit to an empty slot... this next chunk will handle the other case we need to deal with, if the piece is jumping another piece.

```
} else if(%xDistance == 2 && %yDistance == 2)
{
    // Check for jumps
    if(%fromX > %toX)
    {
        %xSign = -1;
    } else
    {
        %xSign = 1;
    }

    if(%fromY > %toY)
    {
        %ySign = -1;
    } else
    {
        %ySign = 1;
    }

    %newX = %fromX + %xSign;
    %newY = %fromY + %ySign;
```

We only check for jumps if the x and y distance is 2. To prepare for our jump checks we check what direction we are heading to get the movement direction modifier we need to check a slot, this modifier will be represented by an x and y value, each being either 1 or -1 (since a piece can only move diagonally neither of the values will be 0). If fromX is greater than toX then we are heading upward, which in our current setup is in the negative direction so we set the xSign to be -1. If we are heading downward we set it to 1. We do this comparison for both x and y that way we can get the middle spot in between our from and to locations (since this has to be a 2 unit move because we're jumping). As you can see we get the newX and newY by adding the x and ySign to the from locations, which again gets us our midpoint, so we can check if there is a valid checker to jump. Our next chunk will do the rest of the jump decisions.

```
// get the spot in between where the client is trying to moveMap
// and where the client's piece came from
%midSpot = %this.getPiece(%newX, %newY);

// here we set a value to check for king's to jump
%king = getKing(%piece);
%normal = getNormalPiece(%piece);

// check if the midspot is either a teammate, a teammate king, or nothing,
// if not then we jump it
if((%midSpot == %normal) || (%midSpot == $none) || (%midSpot == %king))
{
    return false;
} else
{
    return %newX SPC %newY;
```

Torque Game Builder – Checkers Tutorial - Part 4

```
    }  
  }  
  
  return false;  
}
```

Code Sample 4.1.3

Now that we have the logical location of the midSpot between our piece's original location and the location it's trying to move (jump) to we need to get the type of object that's in that slot, if any. So we do a `getPiece()` at that location and store it in `%midSpot`. We then call a function `getKing()` and store it in a variable called `%king`. We still need to create this function, but all the function will do is get the king value of the team of our piece since the regular piece object for a team is stored as `$red` or `$blue` and the king is actually a separate object type like `$redKing` and `$blueKing`. We need to compare it to both the regular type and the king type since we will allow our piece to jump both a regular and a king piece. As you can see we do the same thing as we did with the king piece and call a `getNormal()` function which does the same thing, returns the normal piece. We can now finally do our comparison. We first check to see if the midSpot piece is a normal piece of our same team, or if it's an empty slot, and finally if it's a king piece of our same team. If any of these are true then we aren't allowed to jump it so we return false, else it must be a valid jump so we return the jumped spot. After all of this checking we end with a return false. If it hasn't triggered a return true yet then it must be a false jump. This will happen if you try to make a piece move more than 2 units.

That's it, we have finished our move logic. It is a decent amount of checkers, calculations, and decisions, but with that done the core of our gameplay is ready to be utilized. Before we move on we need to create the `getKing()` and the `getNormal()` functions that we called and have not yet been created, so add these two functions to our checkerboard.cs.

```
function getKing(%team)  
{  
  // check what team is being passed in and return the proper king  
  if(%team == $red)  
  {  
    %king = $redKing;  
  } else if(%team == $blue)  
  {  
    %king = $blueKing;  
  }  
  
  return %king;  
}  
  
function getNormalPiece(%team)  
{  
  // check what team is being passed in and return the proper normal piece  
  if(%team == $red || %team == $redKing)  
  {  
    %piece = $red;  
  } else if(%team == $blue || %team == $blueKing)  
  {  
    %piece = $blue;  
  }  
}
```

Torque Game Builder – Checkers Tutorial - Part 4

```
    return %piece;
}
```

Code Sample 4.1.7

These functions essentially do a very similar thing. They first check what team we're passing in. If the piece is either \$red or a \$redKing then we return the \$redKing if this is the getKing() function or we return a \$red if this is the getNormal() function. If we pass in either a \$blue or \$blueKing we return a \$blueKing if we are calling the getKing() function or a \$blue if we are calling the getNormal() function. This allows us to easily compare teams and pieces.

This pretty much covers our CheckerBoard class. The only thing that we could add is a check for double jumps, which I have in the official checkers demo, though we won't go into that in this tutorial.

Now we can set up our game to be triggered to start properly.

4.2 Triggering our game start

After we establish a connection we want to do a few things. First we want to hide away our Connection Gui and then we want to load up our checkerboard from our level. The first step is rather easy. Since we started by giving that GuiControl a name, we can simply make it invisible.

We need to initiate this when our server has established a connection. There is a useful function that gets called when a client has connected to our server called "onClientConnected()". This is called a "callback" as you might have heard referenced in Torque before. What this means is that we can have this determine if it's a valid connection (not over two people) and then send a command to that client that does what we are wanting to be done after establishing a connection.

We also have another helpful callback called "onServerCreated()". Between the two of these we can create the proper sequence of events. Dig into your "server" folder and open initServer.cs and add this function to it to start this process.

```
function onServerCreated()
{
    // toggle first connection to true
    $firstConnection = true;
}
```

Code Sample 4.2.1

What this does is toggle a global variable on the server when the server is created. We will use this in the onClientConnected() function to determine what client is connecting.

Now add this function after our previous function.

```
function onClientConnected(%client)
{
    // if this is the first connection then its a local connection, second
    // connection is the client connection, any beyond that we send an
    // "in match" message
```

Torque Game Builder – Checkers Tutorial - Part 4

```
if($firstConnection)
{
    $serverConnection = %client;
    setTeam(%client, $red);

    $firstConnection = false;

    initClient();
    initServer();

    Canvas.setContent($MainScreen);
} else if(!$firstConnection)
{
    if($inMatch)
    {
        commandToClient(%client, 'inMatch');
    } else
    {
        setTeam(%client, $blue);
        $playerConnection = %client;

        commandToClient(%client, 'initClient');

        serverLaunchGame();
    }
}
}
```

Code Sample 4.2.2

We do a few things here. First we check if `$firstConnection` is set to true. By default on the first connection this should trigger as true. That means that this connection is the server connecting to itself since it will always be the first connection. It sets the global variable of `$serverConnection` to the client passed in. This `%client` value is an ID that represents the client connection. We then call a `setTeam()` function passing it the client and our global value for red. This function has not been created so we will need to create it. We then set `$firstConnection` to false, that way the next connection will be set as our outside client since our local client has been stored, we continue on by calling an `initClient()` and `initServer()` function. Finally we set our canvas content to `$MainScreen`. This concludes our settings if this is the first connection, our local connection.

Our next comparison checks if `$firstConnection` is set to false. If so then this must some outside client connection. The first thing we do is check to see if an `$inMatch` value is true. The `$inMatch` value will be set when the game launches, that way only two players will connect and any further connections will get a proper message letting them know. If we aren't currently in match we set that client to the blue team, store it as the `$playerConnection`, then we run `initClient`, and finally we launch the game on the server. This will trigger the creation of our checkerboard and the starting of our game since we now have two clients.

Now let's create our `initServer()` and `initClient()` functions. We will place this `initServer()` function at the beginning of this file, since this file is `initServer.cs`.

```
function initServer()
```

Torque Game Builder – Checkers Tutorial - Part 4

```
{
  // Create an object to act as our Server check board
  new t2dStaticSprite(ServerCheckerBoard)
  {
    scenegraph = $checkersScene;
    class = CheckerBoard;
  };

  // init the inmatch value to false
  $inMatch = false;
}
```

Code Sample 4.2.3

This function first creates a new StaticSprite called “ServerCheckerBoard”. We also set its class to CheckerBoard which means it will share the same functions as the CheckerBoard class we created earlier. We end with setting \$inMatch to false. That way it is initialized to be triggered to true when our outside client connections and our game is launched.

The client won't need an initClient function to create an object, since our ClientCheckerBoard will just be the static sprite we already created for our CheckerBoard image in the Level Builder. However, we do need a function to be triggered when the Level Builder loads our client checkerboard, so add this to your “initClient.cs” in your “client” folder.

```
function ClientCheckerBoard::onLevelLoaded(%this, %scenegraph)
{
  $checkersScene = %scenegraph;
}

function initClient()
{
  Canvas.setContent($MainScreen);
}
```

Code Sample 4.2.4

Now we need to create the clientCommand for initClient... so add this function to your “clientCommands.cs” file in your “client” folder.

```
function clientCmdInitClient()
{
  initClient();
}
```

Code Sample 4.2.5

Before we move on there are a couple functions we called in our onClientConnected() that we need to create. The first one is the setTeam() function. We will add this function to our initServer.cs file.

```
function setTeam(%client, %team)
{
  // store the team for the client
  %client.team = %team;
}
```

Torque Game Builder – Checkers Tutorial - Part 4

```
// tell the client what team they are, server is boss
commandToClient(%client, 'setTeam', %team);
}
```

Code Sample 4.2.6

We pass this function our client and the team we want to set that client. We first set a variable on the client called “team” to the team passed. This is for future server reference. We then perform another “commandToClient”. This time it calls another setTeam function passing it the team.

Now we've seen two commandToClient function calls. What commandToClient does is take the client you are calling the function on and it then requires the function name you are calling. Note that this function name must be a **tagged string**. What this means is it uses this ' instead of this “. The name of the function is the only thing that needs to be a tagged string. We then pass any additional values the function will take.

So you can see how we call it on the server side. Now we need to place it on the client side. The client side function needs to be set up a certain way. Since the function name is “setTeam” the client side command will be “clientCmdSetTeam(%team)”... We will add this function to our “client” folder and the “clientCommands.cs” script. We will store all of these specially named functions in the clientCommands and serverCommands script files to separate them from the normal functions. So add this function to your clientCommands.cs file.

```
function clientCmdSetTeam(%team)
{
    // set the player's team
    $playerTeam = %team;
}
```

Code Sample 4.2.7

This is the function that will be called when we do that commandToClient(). It simply takes the %team passed and store it in a global variable on that client called \$playerTeam.

Our onClientConnected also calls another commandToClient... “commandToClient(%client, 'inMatch'); “... which is only called if our server is already in a checkers match. So add this function to the clientCommands.cs file also.

```
function clientCmdInMatch()
{
    // spawn the message telling a client they are rejected, we are in a game without you
    MessageBoxOK("Server in a Match...", "The server you are trying to connect to is already in a match", "");
}
```

Code Sample 4.2.8

This function will simply pop up an OK box notifying the player that the server is already connected in a match.

Now that we have all of our basic game initialization calls handled, it stores the local client as well as the outside client and when it's done it calls “serverLaunchGame()” which will actually launch our game now that our client and server status is already set. Add this function to the empty serverCheckers.cs in the “server” folder.

Torque Game Builder – Checkers Tutorial - Part 4

```
function serverLaunchGame()
{
    // we are no in a match
    $inMatch = true;

    // init the server's boards
    ServerCheckerBoard.init();
    ClientCheckerBoard.init();

    // tell the client to start the game
    commandToClient($playerConnection, 'StartGame');
}
```

Code Sample 4.2.9

This function first sets `$inMatch` to true. It then calls an `init()` function attached to the `ServerCheckerBoard` and `ClientCheckerBoard` (since this is the local connection that has both the server and client on it, it needs to init both). It then ends with a `commandToClient` to the client stored in `$playerConnection` (the outside client) and calls the “StartGame” function. As we know this will call a function named “`clientCmdStartGame()`” on the client, so let's add that function to the `clientCommands.cs`.

```
function clientCmdStartGame()
{
    // call the start game on the client side
    clientLaunchGame();
}
```

Code Sample 4.2.10

All this function does is call a client version of the launch game function. The way I set up the majority of the client and server communication is like this, where the actual `clientCmd` appended function won't do much (unless it's a very simple call or variable setting) but the majority of the functionality will be in a normal function in the `clientCheckers.cs` file. This is the same for the server in the `serverCheckers.cs` file. This makes our game easy to extend and utilize. It also allows us to call the same functionality without using a `clientCmd` if we need. So we will add this new `clientLaunchGame()` function in our `clientCheckers.cs`

```
function clientLaunchGame()
{
    Canvas.setContent($MainScreen);

    // init the client checker board
    ClientCheckerBoard.init();

    // call the game started function on the server
    commandToServer('gameStarted');
}
```

Code Sample 4.2.11

The first thing this function does is canvas content to `$MainScreen` (just like the server side did to the local connection). It then only calls the `init()` method on the `ClientCheckerBoard` since it is not

Torque Game Builder – Checkers Tutorial - Part 4

both a server and a client like our previous function needed. It then ends with a `commandToServer`. This is just like the `commandToClient()` except it doesn't need to specify a server like the client version needs to specify a `%client` since there is only one server per client. It will do the same thing though. It will call a function called "serverCmdGameStarted()" on the server. The `commandToServer()` and `commandToClient()` functions are the core of our turn based networking.

Now we need to create our server command, so add this function to your `serverCommands.cs` in your "server" folder.

```
function serverCmdGameStarted(%client)
{
    // tell the server to begin the game
    serverBeginGame();
}
```

Code Sample 4.2.12

Here we simply call a normal function called `serverBeginGame()`, that way we keep most of our game processing and logic in normal functions. So add this function to your `serverCheckers.cs`.

```
function serverBeginGame()
{
    // set turn to the server's turn
    setTurn($serverConnection);
}
```

Code Sample 4.2.13

We first call the `setTurn()` function passing it our `$serverConnection`. We still need to create this function but it just handles the setting of turns both on the client and the server side. Now we need to create the `setTurn` function, right after our previous function.

```
function setTurn(%player)
{
    // get the other player
    %otherPlayer = getOtherPlayer(%player);

    // call the client's set turn function properly
    commandToClient(%otherPlayer, 'setPlayerTurn', $false);
    commandToClient(%player, 'setPlayerTurn', $true);

    // set the server's turn properly
    $playersTurn = %player;
}
```

Code Sample 4.2.14

This function first gets the `%otherPlayer` using a function called `getOtherPlayer` (we still need to create this function but it will work much like our `getKing()` function works). We then call the `setPlayerTurn` function on the `%otherPlayer` client and then on the `%player` client (the `%player` is the one passed). We end by setting the global variable `$playersTurn` to the player passed.

Torque Game Builder – Checkers Tutorial - Part 4

First let's create the utility `getOtherPlayer()` function, so add this after our previous function.

```
function getOtherPlayer(%player)
{
  // we check to see what play was passed and return the other player
  if(%player == $playerConnection)
  {
    %otherPlayer = $serverConnection;
  } else
  {
    %otherPlayer = $playerConnection;
  }

  return %otherPlayer;
}
```

Code Sample 4.2.15

This basically just compares the player passed and passes the other player back.

Now that we created the server side set turn function we need to create the `clientCmd` function on the client side in the "clientCommands.cs" file in the "client" folder.

```
function clientCmdSetPlayerTurn(%val)
{
  // set the player's turn to the value passed
  $myTurn = %val;
}
```

Code Sample 4.2.16

This stores whether it's this player's turn in a global variable called `$myTurn`, nothing special.

Now we need to backtrack a step and create two functions that we called earlier but skipped past the creation of them... These are the `init()` functions of the `ServerCheckerBoard` and the `ClientCheckerBoard`. We will add the `ServerCheckerBoard.init()` function first in the "serverCheckers.cs" in your "server" folder.

```
function ServerCheckerBoard::init(%this)
{
  // init the team counts
  $redCount = 0;
  $blueCount = 0;

  // loop through and create the server board
  for(%x = 0; %x < 8;%x++)
  {
    for(%y = 0;%y < 8;%y++)
    {
      // check if this is a black checkerboard spot
      if(%this.isBlack( %x, %y ))
      {
        // check if this is a checker's starting spot
```

Torque Game Builder – Checkers Tutorial - Part 4

```
        if(%y < 2)
        {
            // set the the red team's piece and add to the count
            %this.setPiece(%x, %y, $red);
            $redCount++;
        } else if( %y > 5 )
        {
            // set the the blue team's piece and add to the count
            %this.setPiece(%x, %y, $blue);
            $blueCount++;
        } else
        {
            // set the blank spot
            %this.setPiece(%x, %y, $none);
        }
    }
else
{
    // set the blank spot
    %this.setPiece(%x, %y, $none);
}
}
}
```

Code Sample 4.2.17

First we set the \$redCount and \$blueCount to 0. This is so we can increment them as we loop through and add the starting checker pieces for each of those teams. We then start to loop through and create our checkerboard. We have a loop through goes through the x locations from 0-7 and then within each of those loops we go through the y loop from 0-7, which will create our 8x8 checkerboard. In each slot we do a few comparisons to see how we need to set it. First we check if the slot is black. If it isn't black then we don't have to worry about a starting checker being placed in that slot. As you can see at the end of this function in our else we set a blank slot. That happens if the slot isn't black. Next we check if y is less than 2. If it is that means the checker slot is in either row 0 or 1, that means its a starting red piece. If it isn't less than 2 we then check to see if it's greater than 5 which checks if it's in 6 or 7, the last two rows, which would mean its a starting blue piece. We properly increment the team counts when necessary.

All in all this function populates our ServerCheckerBoard quite efficiently. Now our ClientCheckerBoard.init() function is a bit more complicated. The ServerCheckerBoard just needs to store the data and locations of the pieces, while the client checkerboard has to visually create the pieces and images of the checkers game. Before we do the client's checkerboard init() function we will create a couple functions that it can reference to make this process a bit easier. Add these functions to your "clientCheckers.cs" file in your "client" folder.

```
function ClientCheckerBoard::getTilePosition(%this, %x, %y)
{
    // grab the board layer's position
    %tileMapPos = CheckerTileLayer.getPosition();

    // divide the position up into x and y variables
    %tileMapPosX = getWord(%tileMapPos, 0);
    %tileMapPosY = getWord(%tileMapPos, 1);
}
```

Torque Game Builder – Checkers Tutorial - Part 4

```
// grab the board layer's size
%tileMapSize = CheckerTileLayer.getSize();

// divide the size up into x and y variables
%tileMapSizeX = getWord(%tileMapSize, 0);
%tileMapSizeY = getWord(%tileMapSize, 1);

// calculate the start position
%tileMapStartX = %tileMapPosX - (%tileMapSizeX / 2);
%tileMapStartY = %tileMapPosY - (%tileMapSizeY / 2);

// calculate the position and pass it back
%tS = CheckerTileLayer.getTileSize();
%pos = (%tileMapStartX + (%x * %tS)) + %tS/2 SPC (%tileMapStartY + (%y * %tS)) + %tS/2;

return %pos;
}
```

Code Sample 4.2.18

This is a helpful utility function. It looks long and scary but its purpose is rather simple. You pass it a logical tile position for the tilemap that will represent our invisible board and it will return the world position. Add this function, right after our previous function.

```
function ClientCheckerBoard::initPiece(%this, %type)
{
    // check which team the piece is
    if(%type == $red)
    {
        // get the count of current pieces of that color
        %num = %this.redPieces.getCount();

        // create a new piece as a clone of our stored piece
        %piece = $redChecker.clone(true);
        %piece.setName("redPiece" @ %num);

        // add the piece to its color's container object
        %this.redPieces.add(%piece);
    } else if(%type == $blue)
    {
        // get the count of current pieces of that color
        %num = %this.bluePieces.getCount();

        // create a new piece as a clone of our stored piece
        %piece = $blueChecker.clone(true);
        %piece.setName("bluePiece" @ %num);

        // add the piece to its color's container object
        %this.bluePieces.add(%piece);
    }

    // set the piece's size and return it
    %piece.setSize("60 60");
}
```

Torque Game Builder – Checkers Tutorial - Part 4

```
    return %piece;
}
```

Code Sample 4.2.19

This function will be used when we create a checker piece in a slot. Just like on our `ServerCheckerBoard.init()` function, we simply set the position as a `$red` or a `$blue`. On our client's case we have a lot more concerns to take into account, so we have this function. What this function does first is check if its `$red` or `$blue`. Each team has the proper functionality though what it does is virtually the same thing, just different team names and imageMaps. First it gets the number of the current count of that team's amount of checker pieces. It uses that number to set the checker piece's name either "redPiece" or "bluePiece" plus the number of that team's piece it is. So basically we create a checker static sprite and set the proper layer and imageMap, and then add it to "`%this.bluePieces`" or "`%this.redPieces`." This is an object that will be created in the `ClientCheckerBoard.init()` function that will hold all of the pieces on the client side. The function ends by setting the piece's size to 60 by 60 and it returns the newly created image.

Basically that function handles all of the repeated creation of our checker pieces, so we don't have to do it in the `init()` function. Now add this next function, right after our previous function..

```
function ClientCheckerBoard::setClientPiece(%this, %x, %y, %type, %piece)
{
    // if the piece isn't empty we do some extra settings
    if(%type != 0)
    {
        // grab the position of the piece
        %pos = %this.getTilePosition(%x, %y);

        // set the position of the piece
        %piece.setPosition(%pos);
        // set the image
        %this.images[%x,%y] = %piece;
    }

    // set the piece's type
    %this.setPiece(%x, %y, %type);
}
```

Code Sample 4.2.20

This is the client side `setPiece` function. Since we have to set more than the server this function will set the client side-specific needs and then you may notice at the end it calls the same `setPiece` function that is on the Server and `ClientCheckerBoard`. The first thing we do is check if the piece is being set to empty. If it isn't then we need to do a few more things. We get the position of this tile and set the position of the `%piece` passed. We then add the `%piece` to another array that stores the actual image object using the x and y position of the tile. This array is much like the board array we use in the base `CheckerBoard` class though this is to store the imageMap reference. At the end of the function we do the base `CheckerBoards setPiece()` function just like the server, since the client will have an identical simulated checkerboard.

Ok now we are ready to create the `ClientCheckerBoard.init()` function. Add this to your "clientCheckers.cs" file.

Torque Game Builder – Checkers Tutorial - Part 4

```
function ClientCheckerBoard::init(%this)
{
    // create container object's for the team
    %this.redPieces = new SimSet();
    %this.bluePieces = new SimSet();
    // here we loop through and create the checker board and the initial checkers
    for(%x = 0; %x < 8;%x++)
    {
        for(%y = 0;%y < 8;%y++)
        {
            // check if the checker board spot is black
            if(%this.isBlack(%x, %y))
            {
                // here we check to see if we're populating the checker's
                // starting area
                if(%y < 2)
                {
                    // create a new red piece
                    %piece = %this.initPiece($red);

                    // set this piece to this location
                    %this.setClientPiece(%x, %y, $red, %piece);
                } else if( %y > 5 )
                {
                    // create a new blue checker piece
                    %piece = %this.initPiece($blue);

                    // set the piece to this location
                    %this.setClientPiece(%x, %y, $blue, %piece);
                } else
                {
                    // set the empty space
                    %this.setClientPiece(%x, %y, $none);
                }
            }
            else
            {
                %this.setClientPiece(%x, %y, $none);
            }
        }
    }
}
```

Code Sample 4.2.21

This loop is very similar to our server init() board creation loop. The only difference is if we are adding a starting checker piece we call initPiece() to create the image of the piece and we also call setClientPiece (which in turn calls setPiece()) instead of directly calling setPiece(). As you can see creating those helper functions helped this potentially bloated function become much simpler and more efficient.

Torque Game Builder – Checkers Tutorial - Part 4

4.3 Test it!

Now we should be able to fire it up and test it. Fire up your TGB application and hit the play button, then click Start Server and then click on Create Server. This time the Connection GUI should disappear. Now fire up a second instance of TGB and click the play button as well, then click on Join Server, click on Query LAN and click on Join Server on the only one that should appear. Now you should get a confirmation that you connected and you should see the awesome checkerboard loaded with all the starting checker pieces! (as shown in Figure 4.3.1)



Figure 4.3.1