

Supporting Document

Design Considerations

Several design considerations were taken into account. Firstly, all game logic should be on the web service, but latency from web service calls generates game lag. To tackle this issue, player game commands execute web service calls independently from a timer that intermittently drops a shape down one row. All web service calls are asynchronous to improve responsiveness.

Concurrency is addressed through efficiency of design: limited web service requests (one up to every 200ms) and lightweight responses; the largest return of data is a jagged array of type `string` with a size of 12x25. In a release version of the prototype, implementation would be on a mainframe, or similar, to allow massively concurrent players. A pre-compiled DLL was provided for use, however it was felt to be more appropriate to design and implement our own C# game logic project.

There were two options for client design: Ajax and Silverlight. Ajax allows part of a web page to be refreshed independently from the main page and would lend smoother gameplay and a flicker-free experience. Alternatively, a Silverlight front-end also achieves the same and adds keyboard movement controls and a more direct interaction between the player and the web service: the Silverlight code executes within the player's browser, thus any movement commands from the player would not need to be sent first to the ASP.NET server and then the web service. For this reason, a Silverlight client was chosen.

A fully decoupled design is necessary, with the our implementation following the Model-View-Presenter (MVP) design pattern; the game logic acts as the Model, the web service as the Presenter, and the Silverlight client as the View. Game commands are sent from the client to the web service, interpreted and forwarded to the game logic. Return data retraces this path. Such a design allows multiple clients to access the web service without affecting one another. This advantage was exploited during development: test clients enabled debugging and several areas of the application could be written at once.

Other design patterns implemented include; Singleton in the `ScoreController` class; high cohesion in the game logic; Proxy in the client as the client logic interacts with a proxy class generated from the web service; Façade in the web service as it acts as the management layer between the client and the game logic; and Factory in `Board.getRandomShape()` to return a new random shape.

The design specification states that games scores must be encrypted into XML. This was achieved using an RSA encryption algorithm and a public key.

Implementation

The two programmers in the group were most confident with C#, so this would be the language used. This enabled the group to check an individual's work. First, initial planning of the implementation took place that specified high level interaction of tiers and outlined the description of some major game classes.¹

¹ See <http://www.simonstanford.co.uk/tetris/Documents/Draft%20Plan.pdf>

Game Logic

Game architecture relies on the assumption that a jagged array of type `string` is needed by the client to draw the game board. This array represents a snapshot of the game board, depicting the current condition of game. Each co-ordinate in the array can be seen as an individual cell block on the game board, and a hexadecimal colour code in a cell block signifies that it is filled. Each `Shape` subclass has a particular colour and draws its unique hexadecimal colour code onto the array. The client reads these colour codes from the array and directly translates it into an image for the player. On each execution of a shape movement method, the web service scans the array and removes any full rows. Type `string` was used in the prototype for the jagged array for ease and readability, but if the game were to be developed into a full release, a more efficient design could be to use type `byte` or `int`.

Real-time game play is achieved through a timer on the client repeatedly calling the web service `MoveBlockDown()` command every 0.2-1.0 second, which returns an updated array which the client then displays. The commands to move left/right, rotate and drop shape are sent via the player's keyboard to the web service separately from the timer, which again returns an updated array. As mentioned, all web service calls are asynchronous to provide more responsive gameplay.

Each shape is a subclass of the `Shape` base class. This allows polymorphism, so `Board.getRandomShape()` can return a `Shape` subclass using the base class reference which can then be manipulated regardless of the sub class. Inheritance was chosen over interfaces as there is common code for all shape objects to move and draw itself on the board, and inheritance prevents duplication.

High cohesion is achieved through separation of concerns: The `Game` object concerns itself with high level management of the player, board, score and game state. The `Board` object manages the game array by removing complete rows and creating a new `Shape` objects. The `Shape` object moves itself around the board.

Silverlight Client

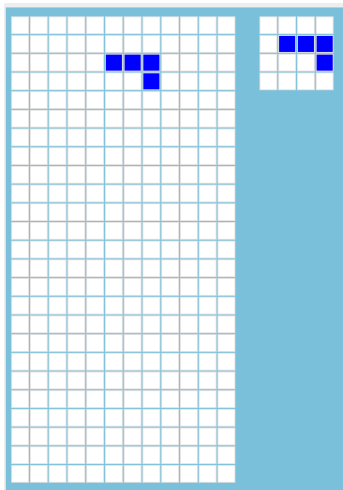


Fig. 1: Collections of Rectangle objects representing the Silverlight game board and next shape board

The client was deployed onto a web site² to allow the complete package to be viewable, and is fully working and playable.

As discussed above, a Silverlight client was chosen due to the potential of direct communication between the client and the web service, and the potential for creating a more playable game. The obvious disadvantage of using Silverlight is that it must be installed on the player's computer and would not work with many tablets or phones. If the client were developed past a prototype, perhaps another framework would be a better choice such as Flash, but this would still require an additional application to be installed on the player's computer.

When a new game starts, the client requests a player name. This is required for logging high scores, so a `RequiredFieldValidator` control is used to ensure a

² See <http://www.simonjstanford.co.uk/tetris>

name is entered. After the player clicks 'Play!' the player's name is forwarded using a query string to the game.aspx page and used to start a new game on the web service. When a new game is started, the first jagged array is posted back from the web service, and this is used to draw the correct number of **Rectangle** objects on the screen. The game then begins.

Through a timer³ or player commands the client connects to the web service and asynchronously executes

its methods. On each execution, the results are returned and the collection of **Rectangle** objects are updated. The hexadecimal colour code at each co-ordinate in the array is then converted⁴ to a **SolidColorBrush** object and the correlating **Rectangle** object is updated.

Each tick of the timer also executes other methods on the web service: `GetGameState()` determines if the game is still playing or not, and `GetScore()` retrieves the current score which is then displayed on the browser window.

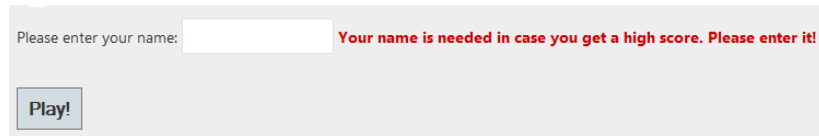


Fig. 2: The RequiredFieldValidator control

High Scores

Player	Score	Date
Simon	320	30/03/2013 19:49:39
Laura	260	31/03/2013 12:37:51
Mike	220	31/03/2013 18:54:57
Dan	160	31/03/2013 10:28:32
Tom	70	30/03/2013 19:26:20
Simon	60	31/03/2013 19:29:59
Harry	50	30/03/2013 19:28:22
Simon	40	30/03/2013 19:30:06
Dick	10	30/03/2013 19:26:56

Fig. 3: High score data retrieved from the web service

The next shape board is an area that displays to the player the next shape that will appear on the board. This information is also requested from the web service on each tick of the timer, and implementation is smaller but identical to the game board. Note that a new shape object in the next shape board is created and placed there, and where needed the reference transferred over to the game board and its co-ordinates changed.

At any time the game can be paused by clicking the pause button, which stops the timer. To prevent the player from being able to move the shape whilst the game is paused, `timer.IsEnabled` is checked on each button press.

When the game is over, `GetGameState` returns false. The client stops the timer and executes the web service method `SubmitScore()`. If the score is a high score, this is then logged in an XML file (see the web service section for details) and a message is displayed. To view the high scores, the player clicks on the 'High Scores' link on the site. This page contains a **GridView** object that is populated with a high scores **DataTable** retrieved from the web service by calling `GetHighScores()`.

³ Microsoft, n.d. *How to: Create a Timer* [online] Available at: <<http://msdn.microsoft.com/en-us/library/cc189084%28v=vs.95%29.aspx>> [Accessed 1 April 2013].

⁴ Stack Overflow, 2011. *How to convert "#00E4FF" to Brush in code?* [online] Available at: <<http://stackoverflow.com/questions/6211388/how-to-convert-00e4ff-to-brush-in-code>> [Accessed 1 April 2013].

Web Service

The web service exposes all the methods necessary to the client for the game to run (see Fig. 4). During development, an initial test method called HelloWorld was created to test communication between the web service and the test client. This was a simple independent method call that purely proved communication between the web service and a client was working.

The web service relies on Session state⁵ to manage concurrent games. This was achieved by placing the `[WebMethod(EnableSession = true)]` attribute at the top of each method. On each command sent to the web service, the game object is retrieved from session state, worked with and then saved and stored back into session state. In this way, many concurrent games can be run and will not interfere with each other. Reduction in traffic between the web service and the client is also achieved, as the client only needs to send the web service a shape movement command to update a game without any additional data such as the current state of the board.

At first, only the MoveBlockDown method returned a game board array. This meant that the board was only updated on every timer tick and made the game lag. As mentioned previously, to rectify this and give a more responsive game all web service game methods were changed to return an updated game board array.

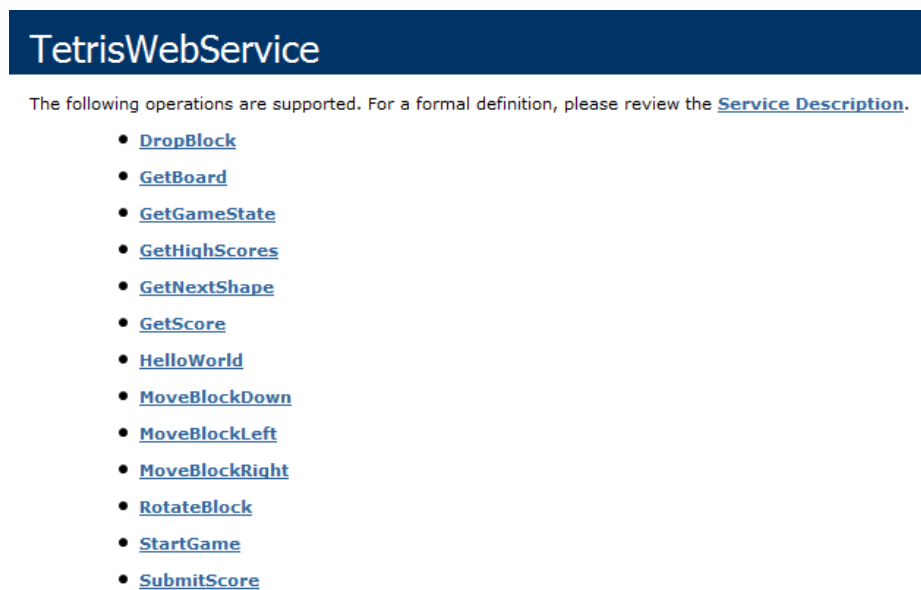


Fig. 4: The unencrypted scores.xml file

To simulate real world use and help demonstration, the web service would be deployed on the internet⁶. This came with its own technical challenges: web user write privileges had to be enabled, to allow new high scores to be written into an XML file, and a client access policy had to be written to permit external web service requests.

⁵ Wellens, S., 2009. Using Session State in a Web Service. *Code Project*, [online] Available at: <<http://www.codeproject.com/Articles/35119/Using-Session-State-in-a-Web-Service>> [Accessed 8 April 2013].

⁶ See <http://www.simonjstanford.co.uk/tetriswebservice/TetrisWebService.aspx>

High Scores

We implemented the high score controller class using the singleton pattern, so that only one instance was available to handle all the requests from the web service. This design ensures data integrity and controlled access to the XML file that stores the top 12 high scores.

```
<?xml version="1.0" encoding="UTF-8"?>
<Scores>
  <HighScore>
    <Score>100</Score>
    <Player>Steve</Player>
    <Date>22/03/2013 22:13:17</Date>
  </HighScore>
  <HighScore>
    <Score>100</Score>
    <Player>Bob</Player>
    <Date>22/03/2013 12:20:20</Date>
  </HighScore>
</Scores>
```

Fig. 5: The unencrypted scores.xml file

The high score controller class uses a custom data structure and an array of scores. This meant we can easily manage the 12 highest scores by seeing how many high scores we had. If we had less than 12 then we add the high score. If we had 12 then we sort the list if the high score is higher than the lowest then we remove the lowest and add the new one.

Once we had this saving and loading the top 12 scores we encrypted the players name to increase the security. We used the Microsoft .Net

System.Security.Cryptography.Xml.EncryptedXml class to encrypt the element (See Fig. 7).⁷⁸⁹

Encryption in this project works using an RSA public and private key pair which are saved in the web.config file. A session key is created separately using the Advanced Encryption Standard (AES) algorithm. This key is then used to encrypt the player name and then the key itself is encrypted using the RSA public key. The encrypted session key and player name is then added to the scores.xml document. Decryption is achieved through using the private RSA key to decrypt the session key, which in turn is used to decrypt the player's name.

Development Testing

Development and testing took place through BitBucket¹⁰, an online distributed version control system (DVCS) that provides a layer of management for team coding and issue tracking. Through this system, the project could be developed and any issues that arose throughout testing were logged, prioritised, discussed and tackled (see Fig. 9).

An ASP.net project was created to test the game logic functions. This initially referenced the Tetris game logic directly and meant issues could

Issues (1–14 of 14)

Title	T	P	Status
#14: Getting double score inserts into the XML file	🔴	📈	RESOLVED
#1: Move encryption key to web config	🔴	📈	RESOLVED
#2: Move scores file location to web config	🔴	📈	RESOLVED
#13: No checking is done to see if shapes can rotate	🔴	📈	RESOLVED
#11: Scores sent using SubmitScore() aren't updating the scores.xml file	🔴	📈	RESOLVED
#12: Game board size	🔵	📈	RESOLVED
#10: GetGameState() returns true if GameOver and false if playing	🔴	📈	RESOLVED
#9: RemoveLine() is shifting the rows up, not down	🔴	📈	RESOLVED
#8: Web service movement calls don't return a value	🔵	📈	RESOLVED
#7: What is the difference between MoveBlockDown() and DropBlock()	🔵	📈	RESOLVED
#6: Next Shape	🔵	📈	RESOLVED
#5: J_Shape does not implement Rotate	🔴	📈	RESOLVED
#4: Array structure a little messed up	🔴	📈	RESOLVED
#3: Use string array instead of int array	🔴	📈	RESOLVED

Fig. 6: BitBucket issue log for game logic & web service

⁷ Microsoft, n.d. *How to: Encrypt XML Elements with Asymmetric Keys* [online] Available at: <<http://msdn.microsoft.com/en-us/library/ms229746.aspx>> [Accessed 5 April 2013].

⁸ Microsoft, n.d. *EncryptedXml Class* [online] Available at: <<http://msdn.microsoft.com/en-gb/library/system.security.cryptography.xml.encryptedxml%28v=vs.80%29.aspx>> [Accessed 5 April 2013].

⁹ Microsoft, n.d. *EncryptedXml Class* [online] Available at: <<http://msdn.microsoft.com/en-gb/library/system.security.cryptography.xml.encryptedxml%28v=vs.100%29.aspx>> [Accessed 5 April 2013].

¹⁰ <https://bitbucket.org/robbleasdale/m32com-assignment-1> & <https://bitbucket.org/sstanford/m32com-assignment-1-tetris-client>

M32COM Tetris Web Service Game

Rob Bleasdale, Michael Edionwe, Remi Hameed & Simon Stanford

be debugged without the complication of the web service. Once the game logic was completed, development of the web service began, with the test project again used for analysis and debugging.

The test project contained three pages; Game.aspx tested game logic including shape rotation, collision detection, complete line removal and game over situations (see Fig. 8); ViewHighScores.aspx was used to test retrieval the high score values and check the decryption of the scores file; SubmitHighScore.aspx simulated adding new score values to the scores.xml file and to test the encryption.

Following completion of the game logic and web service, they were deployed onto a web server. This made the assignment truer to real life, and made development of the Silverlight client easier.

Group Constitution

Rob Bleasdale: Web service, game logic, supporting documentation

Michael Edionwe: Preliminary UI, user testing

Remi Hameed: User testing

Simon Stanford: Silverlight client, game logic, website deployment, supporting documentation

M32COM Tetris Web Service Game

Rob Bleasdale, Michael Edionwe, Remi Hameed & Simon Stanford

```
<?xml version="1.0" encoding="UTF-8"?>
<Scores>
  <HighScore>
    <Score>100</Score>
    <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element" xmlns="http://www.w3.org/2001/04/xmlenc">
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc" />
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
          <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
          <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
            <KeyName>rsaKey</KeyName>
          </KeyInfo>
          <CipherData>
            <CipherValue>enzDxkHIB9s0qrCr0YQX8fh6sSa/1XfCX/ol3IbfffIcJuioY5N7YqEZ/PsexmFAoSJHnVZIptztlv7LQ3Xho6xRmRGUMkJJJa2QsvxDZqzdrCf5zyg50f2g+FbINDJBTKc7v1hYH+Y6x9yTqb1ncMOR5XT9yoDdlwwC24pJDUEA=</CipherValue>
          </CipherData>
        </EncryptedKey>
      </KeyInfo>
      <CipherData>
        <CipherValue>rrrx373q/foihzMbqC/ZJxS7hm6BWCQ5vTNh9hmmtKfkd3ngP9Guzg42RZBsJKLd</CipherValue>
      </CipherData>
    </EncryptedData>
    <Date>22/03/2013 22:13:17</Date>
  </HighScore>
  <HighScore>
    <Score>100</Score>
    <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element" xmlns="http://www.w3.org/2001/04/xmlenc">
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc" />
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
          <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
          <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
            <KeyName>rsaKey</KeyName>
          </KeyInfo>
          <CipherData>
            <CipherValue>kc7v1hYH+Y6x9yTqb1ncMOR/1XfCX/ol3IbfffIcJuioY5N7YqEZ/PsexmFAoSJHnVZIptztlv7LQ3Xho6xRmRGUMkJJJa25XT9yoDdlwwC24enzDxkHIB9s0qrCr0YQX8fh6sSaQsvxDZqzdrCf5zyg50f2g+FbINDJBTKpJDUEA=</CipherValue>
          </CipherData>
        </EncryptedKey>
      </KeyInfo>
      <CipherData>
        <CipherValue>3ngP9Guzg42RZBsJKLdrrx373q/foihzMbqC/ZJxS7hmmmtKfkd3ngP9GvTNh9hmmtKfkd</CipherValue>
      </CipherData>
    </EncryptedData>
    <Date>22/03/2013 12:20:20</Date>
  </HighScore>
</Scores>
```

Fig. 7: The encrypted scores.xml file

M32COM Tetris Web Service Game
Rob Bleasdale, Michael Edionwe, Remi Hameed & Simon Stanford

Name:

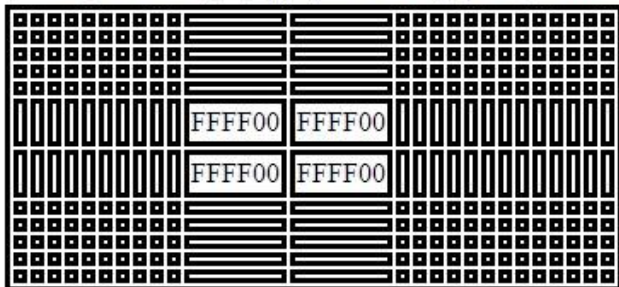


Fig. 8: The test Game.aspx page

Title	Type	Priority	Status	Assignee	Created	Last updated	Actions
#14: Getting double score inserts into the XML file	bug	minor	resolved	unassigned	5 days ago	4 days ago	May have fixed this - because web service calls in Silverlight are asynchronous, I think it ran the game over code twice. I've added a bit of logic to make to try and fix it, looks ok at the moment.
#1: Move encryption key to web config	bug	major	resolved	Robert Bleasdale	13 days ago	5 days ago	fixed in commit 61a1820
#2: Move scores file location to web config	bug	major	resolved	Robert Bleasdale	13 days ago	5 days ago	fixed in commit 61a1820
#13: No checking is done to see if shapes can rotate	bug	major	resolved	unassigned	7 days ago	5 days ago	Old co-ordinates of shape saved and restored if exception occurs, debugging of shape.rotate() methods
#11: Scores sent using SubmitScore() aren't updating the scores.xml file	bug	major	resolved	unassigned	7 days ago	5 days ago	Write permissions set on web server, xml file deleted to remove corrupt encryption key
#12: Game board size	enhancement	trivial	resolved	unassigned	7 days ago	6 days ago	Reduced the height of the array to 25, and it feels a bit more manageable.
#10: GetGameState() returns true if GameOver and false if playing	bug	major	resolved	unassigned	7 days ago	7 days ago	Fixed in commit 5b3770c
#9: RemoveLine() is shifting the rows up, not down	bug	major	resolved	unassigned	8 days ago	8 days ago	Debugging of Board class
#8: Web service movement calls don't return a value	enhancement	minor	resolved	unassigned	9 days ago	9 days ago	All webservice calls return the game board. including the start game.
#7: What is the difference between MoveBlockDown() and DropBlock()	proposal	minor	resolved	unassigned	9 days ago	9 days ago	The drop block is meant to send the block all the way to the bottom so you don't have to wait for it to drop each second.
#6: Next Shape	enhancement	minor	resolved	unassigned	12 days ago	10 days ago	sent as an array similar to the board just as a smaller array. ie 4 by 4
#5: J Shape does not implement Rotate	bug	major	resolved	Robert Bleasdale	13 days ago	10 days ago	Implemented
#4: Array structure a little messed up	bug	major	resolved	unassigned	13 days ago	11 days ago	I've made changes to the client to support jagged arrays. Everything is in sync now and the Shape objects can draw themselves onto the board using board[x][y].
#3: Use string array instead of int array	bug	major	resolved	unassigned	13 days ago	13 days ago	I have changed the array type and removed the get next shape method

Fig. 9: Web Service Game Logic Issue Log