Pason Battle Tetris Coding Contest Specification

Last Updated: November 9, 2012 3:34pm

Table of Contents

Introduction

The time for discussion is over! The supreme chancellor has decreed that all disputes will be solved through a universally accessible Battle Tetris system. In order to level the playing field, the system is centrally hosted and players are free to devise strategies to improve their chance of winning, but all game play is submitted through the same interface.

A number of high profile packages have been developed to automate game play on their owner's behalf. Some focus on being the fastest; others, on the best moves possible. How will your team topple the best of the best?

Contest Structure

Evaluation

The contest itself will be formatted as a standard single elimination tournament. Contestants will be matched against each other and each match will be comprised of several individual games. Winners will be determined in the following manner:

Game Winner:

1. If one player's stack reaches the top of the game board before the game time limit is reached, the other player will be declared the winner.
2. If neither player's stack reaches the top before the time limit on the game is reached, the player with the most points will be declared the winner.
3. In the case of a tie in points, a random coin flip will be used to determine the winner of the game.

Match Winner:

1. A player will be declared the match winner when they win more than half of the total number of games in the match before the match time limit is reached.
2. If the time limit is reached before one player has won a majority of games, the player who has won more games will be the winner.
3. If the number of games won in the match is a tie, the total number of points for each player will be computed and the player with more points will be the winner.
4. In the event of a tie in points, a random coin flip will be used to determine the winner of the match.

Match winners will advance to the next level of the bracket and will be paired with a new opponent.

Scoring

Scoring and levels are computed in the following way:

1. Base scores for completing lines:
    a. 1 line - 100 points
    b. 2 lines - 200 points
    c. 3 lines - 500 points
    d. 4 lines - 800 points
2. There is a scaling factor applied to points based on the level that a game is at.
3. The level of the game is increased when either player crosses certain point thresholds.
4. Both players will advance levels together.
5. The exact details of these factors and thresholds are outside of the scope here, but can be provided upon request.

Key Links and Support

Links

Main Contest Website: http://codingcontest.pason.com/

Forum Website: http://codingcontest.pason.com/forum/

Main Contest Contact Email: pasoncontest@pason.com

Support

We want participants to have fun and be successful developing and competing with their clients. Therefore, some support will be available to assist with problems that may arise. We can't guarantee a particular level or amount of support and may not provide help if we feel that it would give an unfair advantage or detract from the quality of the contest.

The first place to look for help and ask questions is in The Forum. Please start there and, if appropriate, post your question so that others who have the same question can see the answer. Otherwise, please contact us through the channels provided on the main website and we will try to assist you.

Game Overview and Detailed Rules

Overview

The game for the coding contest is based on a tile stacking game called Tetris. The object of the basic game is to stack the pieces by rotating and moving them so that the shapes create full horizontal rows. When a row is completed, it is removed from the game. There is a free basic online version of the game available.

Battle Tetris (online game) is slightly different than basic Tetris in a few different ways:

1. Game play is a competition rather than individual.
2. When one player completes rows based on the rules, opponents have new horizontal rows automatically added to their game board.

Detailed Rules

For the coding contest, the detailed rules are as follows:

1. The game will use the standard tetrominos. See the appendix for more information.
2. Players are able to rotate, move, and drop pieces.
3. Rotation will be carried out around the origin in the tetromino.
4. Pieces will continue to fall down automatically if players do not make moves.
5. The game board will be 10 cells wide and 20 cells high.
6. The origin of the game board is the top left corner at 0, 0.
7. All pieces will enter the game board with their origin set to column 5, row 0.

8. As game levels progress, the natural rate at which pieces fall will increase.
9. There are several rules for sending lines to an opponent:
    a. When a single row is completed, no lines are sent to the opponent.
    b. When two are completed at once, a single line is sent to the opponent.
    c. When three are completed at once, two lines are sent to the opponent.
    d. When four lines are completed at once, four lines are sent to the opponent.
10. The lines that are sent to an opponent will have gaps in them and players may remove them by completing them as normal.
    a. The lines sent to the opponent are generated using a coin flip for each cell in the row. In addition there will be at least 1 cell empty so you will never get a full filled row. You might get a full empty row (although chances are very rare).

    b. The rows are added to the bottom of the board and will shift all existing rows up.
11. Player algorithms will compete against one another in automated games.
12. Each player will have independent pieces from other players.
13. Both players in a game will share a common queue of upcoming pieces.
14. If a client drops a piece as a move, it will receive the next piece in the queue immediately.
15. The system will automatically lock in pieces as they fall naturally; delivery of subsequent pieces is not immediate if tetrominos are locked in this way. The piece will be placed on the game board at the start of the following time step.
    a. If both players lock in a tetromino in the same time step automatically, the order that pieces are dispatched from the shared queue is random.
16. The game is over when the first player exceeds the 20th row of the board with a locked piece.
17. The first player to exceed the 20th row loses.
18. Client code must be executed on the systems provided and cannot be supplemented with additional computational capabilities beyond what is available.

Getting Started

Infrastructure and Virtual Machines

The coding competition this year is going to be held using Amazon Web Services (AWS). Pason will be hosting these systems and participants will be provided with one or both of a username/password combination and ssh pre-shared key as well as a fully qualified domain name (hostname) for their system. In addition, participants will be able to request either a Linux VM using Ubuntu or a Windows VM with Windows Server 2008 R2 (either 32 or 64 bit).

To request a VM, simply email your team name and your VM choice (Linux or Windows) to pasoncontest@pason.com. We will respond to that email with your VM credentials.

Standard Software

Each of the virtual machines will be provided with an initial set of software:

Linux

- Standard installation including development tools (e.g. gcc/g++, make, etc)
- Python will be installed
- Java SDK will be installed
- ZeroMQ will be installed
    - Bindings for C, C++, Java, and Python will be provided/installed

Windows Server 2008 R2

- Python will be installed
- Java SDK will be installed
- ZeroMQ will be installed
    - Bindings for C++, Java, C#, and Python will be provided/installed

ZeroMQ Library and Binding Paths

Linux

ZeroMQ

File(s)	Path
libzmq	/usr/lib

Java

File(s)	Path
zmq.jar	/usr/local/share/java
libjzmq	/usr/local/lib

C

File(s)	Path
libczmq	/usr/local/lib

Windows

ZeroMQ

File(s)	Path
libzmq.dll	C:\Windows\System32

Java

File(s)	Path
zmq.jar	C:\zmq\java\lib
jzmq.dll	C:\zmq\java\lib

C#

File(s)	Path
clrzmq.dll	C:\zmq\c#
clrzmq-ext.dll	C:\zmq\c#

Development Environment

As part of the contest, it is required that game clients run from the AWS server instances but how the code is developed is largely up to the individual teams. Based on some initial testing, it appears that graphical IDEs can be run on the AWS systems but the performance of them is fairly low. An alternative solution may be to develop code on local or other higher performance systems and then move the code to the AWS system. This requires that the code be compiled on the AWS systems using command line tools. In order to simplify that process, a few sample links are provided that demonstrate some of the ways in which the command line tools can be used:

- Linux C/C++ (gcc/g++) Example: http://pages.cs.wisc.edu/~beechung/ref/gcc-intro.html
- Windows C/C++ (cl) Example: http://msdn.microsoft.com/en-us/library/ms235639%28v=vs.80%29.aspx
- Windows C# Example: http://msdn.microsoft.com/en-us/library/vstudio/78f4aasd.aspx
- Windows/Linux Java Example: http://www.sergiy.ca/how-to-compile-and-launch-java-code-from-command-line/

Note: C/C++ development tools for Windows, Java IDEs, and Python IDEs are not provided but participants are free to add them. Here is a list of some sample products available for these purposes that can be downloaded and installed without payment:

- Visual Studio Express 2012 - A graphical IDE for development of C/C++ on Windows systems.
- Eclipse - A graphical IDE primarily used for Java development but has plugins for other languages such as C++ and Python. Runs on most O/Ss.

Accessing VMs

The instructions for accessing each VM is a little bit different depending on which operating systems are on the VM host and the client.

Linux Client - Linux VM

1. Save the ssh key file provided (assume it's named keyfile)

    a. Note: This is a private ssh key. If it is compromised, it will allow full access to your Linux VM.
2. Ensure the permissions on the key file are 0600. (hint: look up chmod)
3. Use ssh (remote shell) for terminal access and scp (secure copy) to copy files with the following sample commands
    a. $ ssh -i keyfile username@hostname
    b. $ scp -i keyfile localfile username@hostname:/path/on/host
    c. $ scp -i keyfile -r localdirectory username@hostname:/path/on/host

Windows Client - Linux VM

1. The main tool that is used to access Linux systems from Windows is Putty. The full Windows installer is the recommended mechanism for obtaining this package.
2. Save the ssh key provided in a file.
3. Once installed, launch Putty.
4. In the left hand pane under Connection->SSH->Auth, there is a field named "Private key file".
5. Click on the 'browse' button to the right and select the file that contains the ssh key.
6. Navigate back to Session in the left hand pane and enter username and hostname information and connect to the remote system with a shell.
7. To transfer files, Putty comes with a PSFTP program.
    a. A tutorial is available for PSFTP.

Windows Client - Windows VM

1. Use **Remote Desktop Connection** with the provided username and password to access the system.
    a. There is a file transfer mechanism provided with remote desktop for moving files.

Linux Client - Windows VM

1. There are a wide range of packages available that work with Windows Remote Desktop but they vary by distribution. Locate and use the one for your distribution with the provided username and password.

X11 Tunneling

For those that are interested, X11 will be available on the Linux VMs and it's possible to use graphical programs from the remote VM system on local machines. There can be some nuance and difficulty getting this to work properly but, often, it can be accomplished using default configurations.

From a Linux client when connecting with ssh, add the '-X' option to the command to invoke the X tunnel. On Windows, Putty supports the X11 tunnel but there is an additional X Window client that is required: XMing seems to work fairly well most of the time but others are also available.

Given the complexity of this area and the wide variety of problems that can result, we may not be able to provide support troubleshoot X tunneling issues if they should arise.

Client Outline and Sequence Information

These are the approximate steps and activities for a basic game client. There are many ways this can be written and the exact implementation varies by development language. However, the general structure and calling sequence is correct.

```
// Create connections to central game system
// Make sure to use the envelope for pub/sub
Establish pub/sub ZeroMQ connection to central game system (State Channel)
Establish request/response ZeroMQ connection to the central game system (Command
Channel)

// Connect to the match and obtain client token
// Note: The match token is provided outside of these channels
Frame connect message
Send connect message over command channel
Receive and store client token

// Wait for game to start
Monitor state channel for game board state matching this game

// Play the game (intentionally vague)
Monitor the state channel for information
Determine moves for each piece
Frame game move messages and send over command channel
```

Game Servers and Match Tokens

ZeroMQ connections to the game system (detailed in the Server API section) require knowledge of what game server to connect to. For this contest, it's possible that the server hostnames can change over time or more systems may be created to handle load as required. In addition, each match is identified by a match token that is required to tell the server what match you're attempting to connect to. On game day, both of these fundamental pieces of information (the IP/hostname of the game system and the match token for your match) will be provided shortly before a match takes place.

Testing

There are two ways to create test matches.

1. Test matches can be created manually on the Game Testing tab of the Team Information website page.
    a. Once a test match has been created this page will show both the match token and the hostname of the game server for the test game.
    b. This manual copy/paste of the match token and server hostname emulates how you will join your contest matches on game day.
2. Submit a GET request using the following URL and format, replacing TEAM_NAME with your team name and PASSWORD with your PASSWORD.
    a. You will be REQUIRED to enter the match token and server hostname manually on game day. Using this URL on game day will continue to create test matches for yourself and will not connect you to your game day matches.
    b. Be sure to URI encode your team name and password otherwise the request will fail.
    c. The response will be JSON.

http://codingcontest.pason.com/scheduler/create_unauthenticated_test_match?team_name=TEAM_NAME&password=PASSWORD

- If the team name and password are correct and a match was successfully created, the response will be of the format:

```
{
    "status": "success",
    "team_name": "team",
    "server_ip": "ec2-30-19-184-123.compute-1.amazonaws.com",
    "match_token": "8b463b895-58cn-4495-a821-0b2dbe2ece06"
}
```

- If an incorrect team name or password is received or the match creation failed, the response will be of the format:

```
{
    "status": "failed",
    "reason": "the reason"
}
```

Server API Specification

Basic Communication With Game System

The interaction between the game playing clients and the game server take place over several network connections.

General Communication

Game connections are implemented using ZeroMQ. ZeroMQ is a cross language communication channel that provides a variety of interconnect

styles (pub/sub, req/resp, etc).

    - ZeroMQ is the only channel that can be used to communicate with the game system.
    - The coding contest is using version **2.1.11** which can be downloaded from the historical releases page. Other versions are not recommended.
- The VMs provided will have ZeroMQ already installed on them.
    - This toolkit varies by language and operating system.
    - There may be several steps to installation (e.g. a base system plus a language binding).
    - A list of the language bindings and instructions is available.
- The documentation is fairly reasonable and there are tutorials available.
    - The Guide has a good introduction and contains examples in many languages.
    - The ZeroMQ API is also available.

Communication with the game system takes place over two distinct channels.

a. The Command Channel is used to send information to the game system such as joining a match and game moves.
b. Commands on this channel return a response to indicate whether the call succeeded or not but do not provide detailed information about the game.
c. The State Channel is where the server will provide information about the state of the game such as game board layout, queue of upcoming pieces, etc.
d. This channel should be treated as read only. The server will ignore information sent on this channel.

- Note: These two channels are independent. They may reside on the same server identified by IP/Hostname or not. Clients should be coded such that they can accommodate the two channels on different endpoints.

Matches and games have some special handling that is worth noting:

- There are no messages to indicate the start of a game other than the server will start providing game state information over the command channel.
- Matches are treated as a set of games and are played back to back.
- When a game in a match ends, the Game End Message will be provided, the server will wait a short period of time, and the next game will automatically begin.
- Any messages received after a Game End Message can be understood as belonging to the subsequent game.
- When the match ends, the Match End Message will be sent and no more communication can be expected from the server for that particular match.

Command Channel

The command channel is used to send information from the client to the server and provide replies directly from the game system. This includes activities like joining matches and making moves.

Key Characteristics:

1. This is a request/response ZeroMQ channel
2. This channel is located on port 5557 and uses TCP
3. The messages that can be sent and expected responses are detailed in this section.
    a. A pipe '|' between strings denotes the values that the variable may take. Only one is allowed.
    b. Once your client is provided with a client token on connect, all further messages must include this token.
    c. The token is a UUID. There is no processing required by the client on it and it can simply be treated as a string.
    d. The sample messages below contain the placeholder token "dc8d75a6-448d-4108-8bfa-52470b97f35f". This will need to be replaced with the actual token that a client receives rather than this placeholder.
4. The message format is JSON.
    a. The line breaks and indentation in the sample messages below is not important but quotation marks, colons, commas, and braces are.

Note: For the messages detailed below, the following conditions and requirements apply:

- Client Command Messages must contain all of the fields detailed within a request message
- Server Response Messages can be expected to contain all of the fields listed.
- Unless otherwise noted for a particular message, the information provided in the message must match exactly.
- This includes case, white space, and content.

Error Message

In all cases when a call to the game system fails, it will respond with an error message. Although all of the commands have an associated server response message, this message may be provided in lieu of those responses in case of an error.

Server Response Message

```
{
    "comm_type" : "ErrorResp",
    "error"     : "ErrorType",
    "message"   : "Error description"
}
```

error - This field will contain a string with the error type.

message - Contains a description of the error that has occurred.

Match Connect

This message is used to establish an initial connection to the game system and acquire a client token.

Note: The match token is not obtained through the ZeroMQ interface. On game day, match tokens will be provided shortly before the match to allow for tracking and to ensure that matches are correctly organized between clients. Tokens for test matches can be obtained through the web site prior to game day.

Note: The client token that is returned must be used in all future communications.

Client Command Message

```
{
    "comm_type" : "MatchConnect",
    "match_token" : "abcd.....",
    "team_name" : "A name"
    "password" : "5up3r53cr3t"
}
```

match_token - This token must be provided by your client and correspond to a valid game.

team_name - This is a name that you provided for your team during registration.

password - This is the password for your team and the one included above is simply an example.

Server Response Message

```
{
    "comm_type" : "MatchConnectResp",
    "resp" : "ok",
    "client_token" : "dc8d75a6-448d-4108-8bfa-52470b97f35f"
}
```

client_token - This token is returned from the call to connect and is required to be provided to most other commands that are issued to the server.

Game Move

Once a game has started, the client sends Game Move messages to initiate movements. When the server gets a Game Move message, the move is executed immediately.

Only one move is allowed per Game Move command and the server will accept any correctly formatted move. **Note:** The server will accept (but will not perform) moves that would violate collision conditions and not report any problem. For example, a client can attempt to move a piece off of the board and the server will accept, but not perform, the move.

Client Command Message

```
{
    "comm_type" : "GameMove",
    "client_token" : "dc8d75a6-448d-4108-8bfa-52470b97f35f",
    "move" : "left"|"right"|"down"|"drop"|"lrotate"|"rrotate"
}
```

move - The move command allows for one, and only one, of the noted options to be sent in a single command. When pieces are dropped, the next new piece is available immediately.

Server Response Message

```
{
    "comm_type" : "GameMoveResp",
    "resp" : "ok"
}
```

State Channel

The state channel is used as a broadcast mechanism to provide details about the game back to the players and other interested parties (such as monitoring applications). Examples of the data on this channel include the current game board, upcoming pieces, and the pieces currently in play.

Key Characteristics:

1. This is a publish/subscribe ZeroMQ channel.
2. The match_token is used as an envelope filter to limit the messages only to those for that particular match.
3. Each message from the publisher comes in two parts:
    a. The first part contains the match_token.
    b. The second part is the JSON message itself.
4. Clients need to read both parts of the message individually and discard the match_token message.
5. This channel is located on port 5556 and uses TCP
6. It is not expected that game clients will try to send any data to the system on this channel.
    a. The messages that can be received from the system are detailed in this section.
7. The messages provided from the system include information for all match clients and it is the responsibility of each client to determine which information is theirs based on the data provided.
    a. Provided that the match_token is properly used for filtering, the messages provided will be limited to that particular match.
    b. Please see the notes in this list regarding the publish/subscribe ZeroMQ channel and filtering.
    c. Ordering of data by client is not guaranteed from one message to the next. That is, the data in an array may be received in the order client2, client1 in the first message but client1, client2 in the next.
8. There are several scenarios in standard Tetris game play in which a particular client may not have a piece on the board.
    a. Game piece information in these cases is either represented as -1 or null. Further details are provided in the specific messages below and clients are responsible for handling these situations.
    b. All messages sent from the game system on this channel will have a monotonically increasing sequence number included that is incremented when any message type is sent.
9. All messages sent from the game system will provide a game server time stamp in milliseconds.
    a. The message format is JSON.
    b. The line breaks and indentation in the sample messages below is not important but quotation marks, colons, commas, and braces are.

Game Board State

These messages are intended to provide information regarding the game board. It is sent when the game board status changes, such as when a piece is locked into position by any of the players in the game.

Server Response Message

```
{
    "comm_type" : "GameBoardState",
    "game_name" : "My New Game",
    "sequence" : "1",
    "timestamp" : 1349298294.465858,
    "states":
    {
        "client1":
        {
            "board_state" :
"00000000000000000000000000000000000000000000000000",
            "piece_number" : 3,
            "cleared_rows" : [7,8,9]
        },
        "client2":
        {
            "board_state" :
"00000000000000000000000000000000000000000000000000",
            "piece_number" : 4,
            "cleared_rows" : []
        }
    }
}
```

states - Contains a representation of the board state for each client. Data is stored as a dictionary with the client name as the key. The board information is interpreted in the following manner:

board_state:

- It is a hexadecimal string.
- There is one bit for each cell in the game board (20 rows * 10 columns = 200 cells / 4 bits per hex value = 50 hex digits).
- The first bit in the data is the top left corner of the board and the following bits increment across the row and then down to the next row.
    - The data is bit packed and there is no spacing or termination from one row to the next. I.e., the first 10 bits are row 0, the second 10 are row 2, etc.
- It only contains data for locked pieces. Pieces that are in play, such as those currently being controlled by the clients, are not included.

piece_number - The current piece number for that client. **Note:** This will contain -1 if the particular client does not currently have a game piece.

cleared_rows - This is an array listing any rows that are cleared at the time of the message.

Game Piece State

These messages provide information about the current piece that each player has in play, including its type and position.

Server Response Message

```
{
    "comm_type" : "GamePieceState",
    "game_name" : "My New Game",
    "sequence" : "2",
    "timestamp" : 1349298294.465858,
    "states" :
    {
        "client1":
        {
            "orient" : 3,
            "piece" : "T",
            "number" : 3,
            "row" : 1,
            "col" : 2
        },
        "client2":
        {
            "orient" : 1,
            "piece" : "I",
            "number" : 4,
            "row" : 4,
            "col" : 2
        }
    },
    "queue": [ "Z",
               "I",
               "Z",
               "J",
               "L"
             ]
}
```

states - This is a dictionary of piece information for each team. For simplicity, the "team_name" provided during team registration is used as the index key and the value is an object containing the piece information. In the sample above, the client names have been replaced with client1 and client2. **Note:** In the event that a client does not have a current piece, the entry for that client will be null. For example if client1 had no piece, the states entry would be "client1" : null.

queue - This is the list of upcoming pieces in the order they will be available. It does not provide any information about initial orientation but simply which piece it will be. Also, note that the the queue is shared between the both players and pieces are distributed on a first come, first served basis.

number - Each of the client states contains a "number" entry. This entry is used to indicate when a new piece has been added.

Game End Message

This message is provided when the system determines that the game has ended.

Server Response Message

```
{
    "comm_type": "GameEnd",
    "game_name": "My New Game",
    "sequence": 352,
    "timestamp": 1350082490.358995,
    "scores": {
        "client1": 0,
        "client2": 0
    },
    "winner": "client2",
    "match_token": "d0d39280-5b95-45ee-aa27-69946547118d"
}
```

scores - A set of the scores for each team indexed by team name.

winner - The team name of the client that won the match.

Match End Message

This message is provided when the system determines that the match has ended.

Server Response Message

```
{
    "comm_type": "MatchEnd",
    "match_name": "Name Of Match",
    "match_token": "d0d39280-5b95-45ee-aa27-69946547118d",
    "status" : "ENDED"
}
```

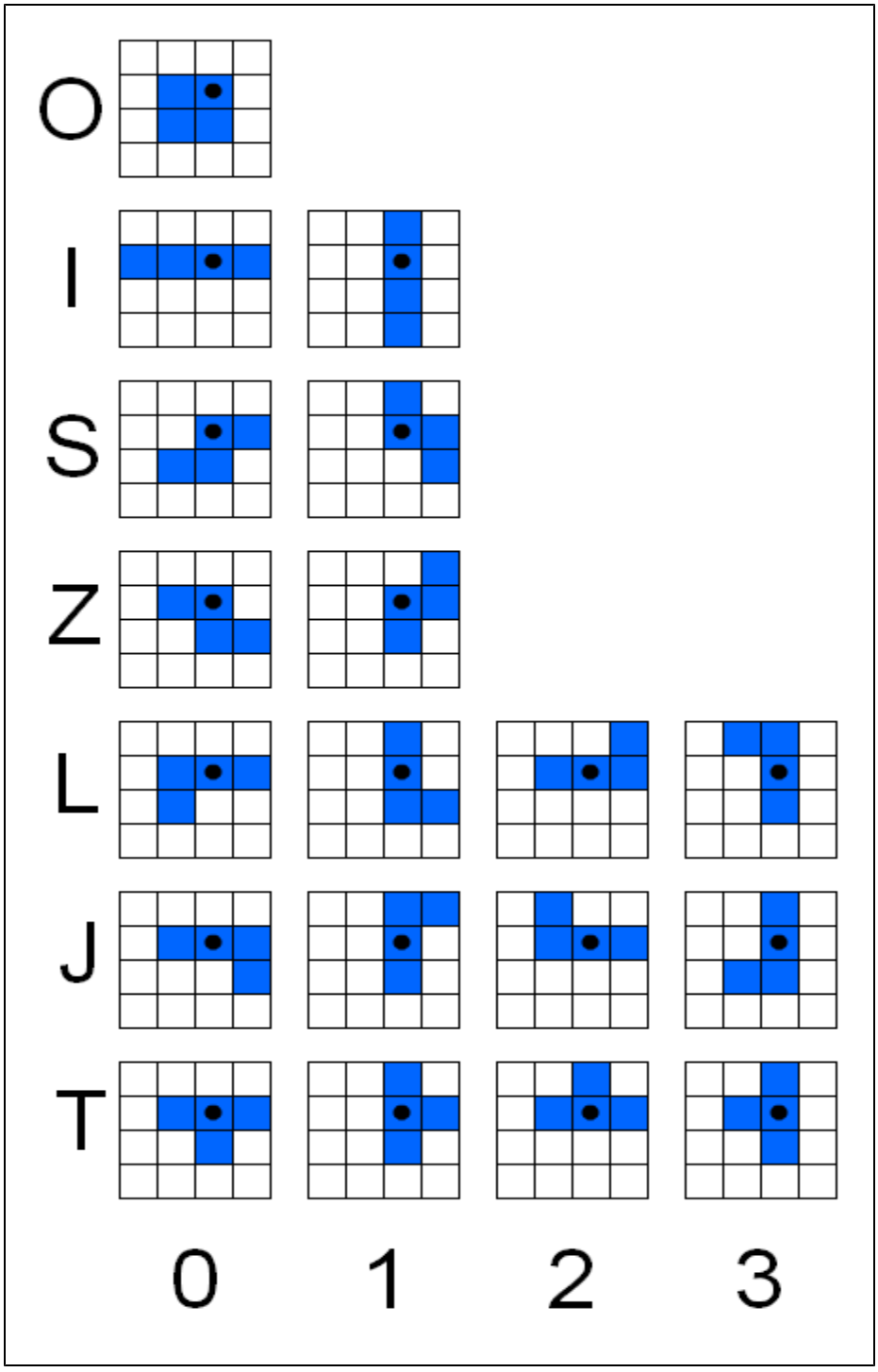
match_name - The name of the match.

match_token - The match token for the match.

status - The status of the match.

Appendix: Tetrominos

There are seven (7) standard tetrominos in the Tetris game. Each piece consists of four squares connected on adjacent edges to form a shape. The figure below has a picture of each of the shapes, the name (just to the left of the image) as well as the progression of the shape as it is rotated. Under each column, there is a number indicating the orientation for that column of pieces.



There are a few important items to note in this image:

a. The black dot in each image is the origin for position in each tetromino.
b. The number of orientations available for each tetromino varies.

2. The leftmost tetromino is in orientation '0', and the orientation number increases by one for each image to the right.
    a. Rotation can be considered as transitioning one image right or left from the current one for a particular shape.
    b. The leftmost image wraps to the rightmost and vice-versa.