PUGCHALLENGE AMERICAS EXCHANGE

2017 Programmer’s Challenge:

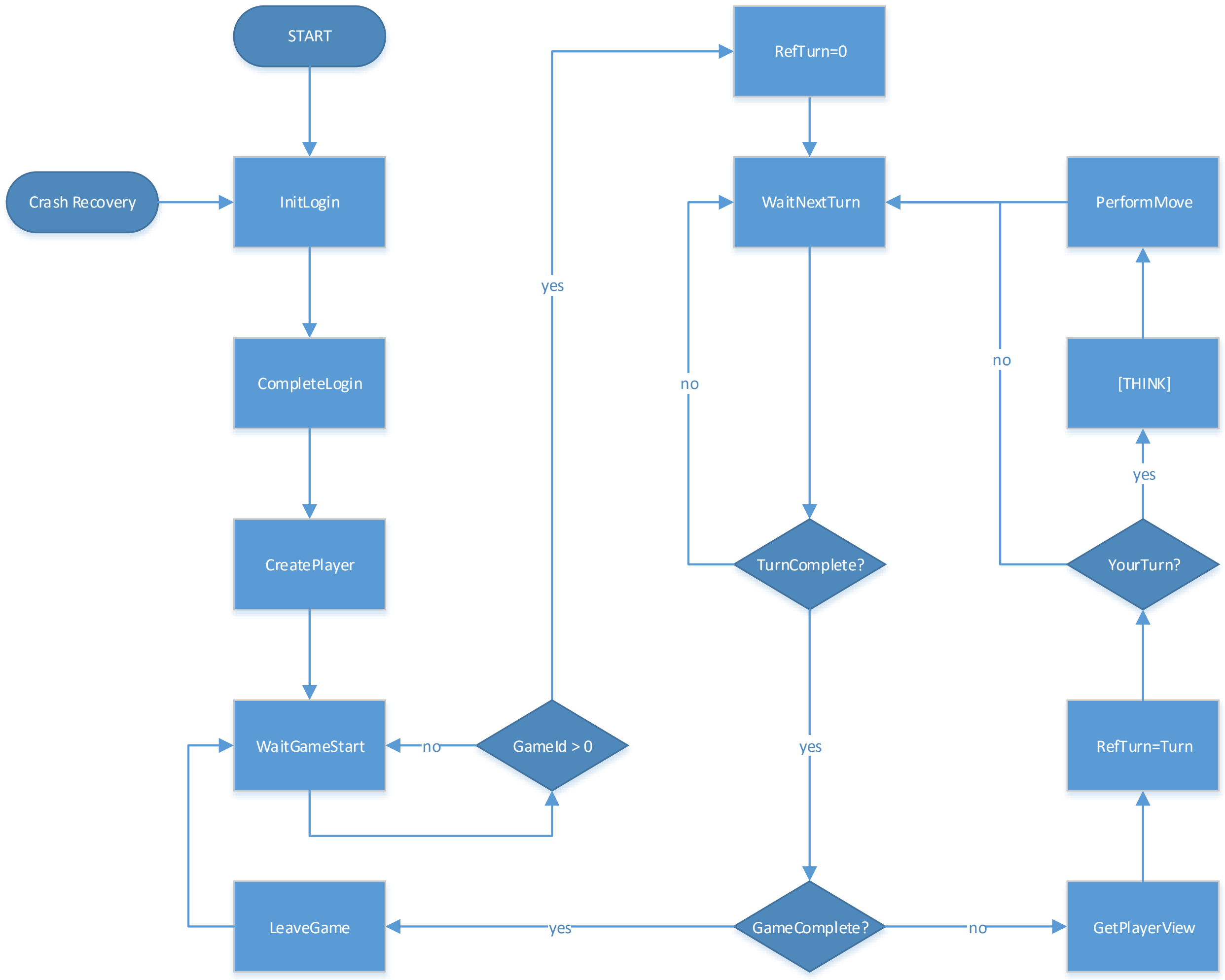
Othello Game

API Documentation

Contents

Application Main Loop

The client should keep run for an entire game, and follow the application flow.



Interfaces

Protocol

The game server uses HTTP and JSON for communication. All requests are made using the HTTP **`POST`** method and all content has Content-Type of **`application/json`**.

Common request data

All requests are made using the HTTP **`POST`** method and all content has Content-Type of **`application/json`** (ie the request is made using JSON in the message body).

All requests must contain an **`Auth`** property with authentication data as described below.

Common response data

All responses are returned with a Content-Type of **`application/json`** (ie the response contains JSON in the message body).

All responses contain two properties:

1. *Message* : this is a string that contains an error message if an error occured during request processing. It will be null otherwise.
2. *Status* : A value indicating whether the request was processed successfully. A status field is returned from the server for all requests. Possible values for this field are in the table

Table 1: Status code values

Code	Value
OK	method executed successfully
WAIT	wait for next turn
AUTH	Authentication error
FAIL	other errors

Authentication

All requests must be authentificated. Authentication information must be sent as part of the request payload in JSON form.

```
"Auth": {
// AuthCode - simple Authentication code, calculated as per this API doc
    "AuthCode" : [string],

// ClientName - a unique (player) name that within a team, provided by you
// (you can use this field to identify several your application instances).
    "ClientName" : [string],

// SequenceNumber - each request should have unique sequence number
    "SequenceNumber" : [int],
// SessionId – during login process gathered id used during whole game
// tournament.
    "SessionId" : [int],

// TeamName – unique team name as you are registered in tournament.
    "TeamName" : [string]
  }
```

AuthCode calculation

The *AuthCode* value is calculated using following algorithm and functions:

```
/* ***************  Function/Procedure Definitions  *************** */
function GetAuthCode returns character (input authString as character):
    define variable utf8String as character no-undo.
    // simple authentication code
    define variable authCode as character no-undo.
    // the team password (from registration)
    define variable pw as character no-undo.

    assign utf8String = codepage-convert(authString + pw,
                                         session:charset,
                                         'UTF-8':u)

           authCode =  lc(
                        hex-encode(
                            sha1-digest(
                                utf8String)))
           .

    return authCode.
end function.


/* **********************  Main Block  ********************** */
define variable authCode as character no-undo.
// team name per registration. unique team name as you are registered in
// tournament.
define variable teamName as character no-undo.
// a unique (per game) player name
define variable playerName as character no-undo.
// the session id. during login process gathered id used during whole game
// tournament.
define variable sessionId as integer no-undo.
// each request should have unique sequence number; recomend use of counter
// for requests
define variable seqNo as integer no-undo.

assign authCode = GetAuthCode(substitute('&1:&2:&3:&4':u,
                            teamName, playerName, sessionId, seqNo)).

/* eof */
```

Initiate session

To get *SessionId*, your application should call *InitLogin* and *CompleteLogin* methods within defined period of time. Fields SessionId and SequenceNumber of Auth object during both method calls must be equal 0.

During *InitLogin* call *Challenge* code is returned. Gathered code should be transformend and used in *CompleteLogin* request. Only both methods calls in short period of time ensure your application participation in tournament.

The challenge code is transformed to a ChallengeRequestCode using the algorithm/function above.

```
define variable challengeRequestCode as character no-undo.
define variable challenge as character no-undo.

assign challenge = '':u // from InitLogin payload
       challengeRequestCode = GetAuthCode(GetAuthCode(challenge)).
```

After method *CompleteLogin* call with proper ChallengeRequestCode SessionId will be returned. SessionId is to be used during all other requests processing.

NOTE: there are should be short period of time between *InitLogin* and *CompleteLogin* calls

Player initialisation

In order to play the game you need to create a player instance. The *CreatePlayer* method is used for this. Each player can participate in only one game at a time.

Request data

There are no extra properties / data request other than the common request data.

Response data

The response will contain the common response data and an INTEGER *PlayerId* field which is used by the server to identify your *TeamName / ClientName* combination.

Waiting for game

Most of the client's time is spent waiting for game. To idle and wait for game *WaitGameStart* is called and results are tracked. *WaitGameStart* returns a *GameId* value*WaitGameStart* have longer response, so there is no need to wait some time between this procedure calls.

Request data

In addition to the common request data, the API expects a *PlayerId* property, containing your assigned Player id (from *CreatePlayer* call).

Response data

In addition to the common response data, the API returns a *GameId* property, containing the current game a player is participating in. If the returned *GameId* equals -1, the game has not started yet and that the player should wait until it does. Any other integer value indicates that game has started.

Wait for next turn

Once a game has started, the application should switch into the move processing loop (see diagram above). In beginning of game and between turns synchronisation is required. For such purposes *WaitNextTurn* method is used.

Request data

In addition to the common request data, the API expects

- *PlayerId* – the value returned from during *CreatePlayer* method call.
- *RefTurn* – the turn number; this is the turn which was completed on the previous time. In the begin of each game or just right after application start this field should contain

0. Each next turn *RefTurn* counter should contain value returned by *GetPlayerView* call.

Response data

The *WaitNextTurn* API waits till the turn completes or a timeout reached. In addition to the common response data, the API returns to following fields

- *TurnComplete* – LOGICAL value. If the turn is not completed during defined period then the *TurnComplete* field contains *false*. You should call method *WaitNextTurn* until you reach *TurnComplete* equal *true.*
- *GameFinished* – *true*, if the game is completed. You should leave game using the *LeaveGame* API and wait for next game – return to game wait loop.
- *FinishCondition* – if game is already completed this filed will show you how: WON – you have win the game, LOST – you lost the game, DRAW – you and opponent got same amount of points.
- *FinishComment* – if you LOST the game, the reason why.
- *YourTurn* – LOGICAL value; shows it it is your turn or not. If it is your turn (YourTurn = true) then you should call *PrepareMove* method during this turn.

Retriving game info

To get information about the current game, including a map of the board, call the *GetPlayerView* API.

Request data

In addition to the common request data, the API expects a *PlayerId* property, containing your assigned player id (from *CreatePlayer* call).

Response data

In addition to the common response data, the following fields are returned

- *Index* – your ID in this game (zero based INTEGER).
- *GameState* –the current game state. One of one of following character values: PLAY, PAUSE, FINISH
- *Turn* – an INTEGER representing your current turn (used as a *RefTurn* value during the *WaitNextTurn* call).
- *PlayerStates* – an array of all participants‘ states. Each array entry is a name/value pair consisting of
  - *Condition* – The current player state (PLAY, WON, LOST, DRAW)
  - *Comment* – Optionally more details
- *Map* – the current game board, including all disk positions.
  - *Width, Height* – INTEGER values representing the current board size.
  - *Rows* – an array containing *Height* rows of *Width* characters. Each character represents a position on the board
    - * represents an empty cell

    - 0 represents your disks
    - 1 represents your opponent‘s disks

PerformMove method

When the *WaitNextTurn* API indicates that it is your turn (via the *YourTurn* field), you should callthe *PerformMove* API. If situation there you do not have valid moves appear you should skip this move by caling *PerformMove* with *Pass=true* specified. If you do not skip moving, so it is the one of possible LOST conditions.

IMPORTANT NOTE: the game server uses zero-based indexes (ie top left cell is {0, 0}

Request data

In addition to the common request data, the API expects

- *PlayerId* – your assigned player ID
- *Turn* – your current move. A pair of Row and Col INTEGER values representing your move
- *Pass* – LOGICAL value ; TRUE if you want to pass/skip this move. If TRUE then the Row/Col values for your *Turn* must be 0, 0

Response data

There are no values returned in addition to the common response data.

Leave game

Once the game completes (*WaitNextTurn* returns *GameFinished=true*), the player must leave the game. This done by *LeaveGame* method call.

This method should be called to ensure which players are visible for server for the next game. If it is not then application will be disconnected from server and team will receive a technical LOST condition.

Request data

In addition to the common request data, the API expects

- *PlayerId* – your assigned player ID

Response data

There is no extra data returned with the common response data.

API description

Method	InitLogin
Call link	/ClientService.svc/json/InitLogin
HTTP method	POST

Request

```
{
  "Auth" : {
    "AuthCode" : [string],
    "ClientName" : [string],
    "SequenceNumber" : [int],
    "SessionId" : [int],
    "TeamName" : [string]
  }
}
```

Request examples

```
{
  "Auth" : {
    "AuthCode" : "9b433ba31796f2db8194644a48cf997d723ca765",
    "ClientName" : "Player1",
    "SequenceNumber" : 0,
    "SessionId" : 0,
    "TeamName" : "Auth"
  }
}
```

Response

```
{
  "Challenge" : [string],
  "Message" : [string],
  "Status" : [string]
}
```

Response examples

```
{
  "Challenge" : "4cd8b50e-e27c-458b-84f1-acb7181e18c4",
  "Message" : null,
  "Status" : "OK"
}


{
  "Challenge" : null,
  "Message" : "GameLogic.AuthException: For login calls,
SessionId and SequenceNumber must be zero.",
  "Status" : "AUTH"

}
```

Method	CompleteLogin
Call link	/ClientService.svc/json/CompleteLogin
HTTP method	POST

Request	`{`
	`  "ChallengeResponse" : [string],`
	`  "Auth" : {`
	`    "AuthCode" : [string],`
	`    "ClientName" : [string],`
	`    "SequenceNumber" : [int],`
	`    "SessionId" : [int],`
	`    "TeamName" : [string]`
	`  }`
	`}`

Request example	`{`
	`  "ChallengeResponse" :`
	`"5c049b982770aeeea03151a34bb48e715fccc752",`
	`  "Auth" : {`
	`    "AuthCode":"9b433ba31796f2db8194644a48cf997d723ca765",`
	`    "ClientName":"Player1",`
	`    "SequenceNumber":0,`
	`    "SessionId":0,`
	`    "TeamName":"Auth"`
	`  }`
	`}`

Response	`{`
	`  "SessionId" : [int],`
	`  "Message" : [string],`
	`  "Status" : [string]`
	`}`

Response examples	`{`
	`  "SessionId" : 567899597,`
	`  "Message" : null,`
	`  "Status" : "OK"`
	`}`
	
	`{`
	`  "SessionId" : 0,`
	`  "Message" : "GameLogic.AuthException: No outstanding`
	`challenge for this client. Init login first.",`
	`  "Status" : "AUTH"`
	`}`

Method	CreatePlayer
Call link	/ClientService.svc/json/CreatePlayer
HTTP method	POST
Request	`{ "Auth":{ "AuthCode" : [string], "ClientName" : [string], "SequenceNumber" : [int], "SessionId" : [int], "TeamName" : [string] } }`
Request example	`{ "Auth":{ "AuthCode" : "4ae5df6334e1d016dcee781d1f24368130bfdcee", "ClientName" : "Player1", "SequenceNumber" : 1, "SessionId" : 567899597, "TeamName" : "Auth" } }`
Response	`{ "PlayerId" : [int], "Message" : [string], "Status" : [string] }`
Response examples	`{ "PlayerId" : 1, "Message" : null, "Status" : "OK" }` `{ "PlayerId" : 0, "Message" : "GameLogic.AuthException: Stale session id. Relogin or stop.", "Status" : "AUTH" }`

Method	WaitGameStart
Call link	/ClientService.svc/json/WaitGameStart
HTTP method	POST

Request	`{`
	`  "PlayerId" : [int],`
	`  "Auth" : {`
	`    "AuthCode" : [string],`
	`    "ClientName" : [string],`
	`    "SequenceNumber" : [int],`
	`    "SessionId" : [int],`
	`    "TeamName" : [string]`
	`  }`
	`}`

Request example	`{`
	`  "PlayerId" : 1,`
	`  "Auth" : {`
	`    "AuthCode" : "425262ba0a44c87184220a8b5509e6fac5d3fa3f",`
	`    "ClientName" : "Player1",`
	`    "SequenceNumber" : 2,`
	`    "SessionId" : 567899597,`
	`    "TeamName" : "Auth"`
	`  }`
	`}`

Response	`{`
	`  "GameId" : [int],`
	`  "Message" : [string],`
	`  "Status" : [string]`
	`}`

Response examples	`{`
	`  "GameId" : -1,`
	`  "Message" : null,`
	`  "Status" : "OK"`
	`}`

Method	WaitNextTurn
Call link	/ClientService.svc/json/WaitNextTurn
HTTP method	POST

Request

```
{
  "PlayerId" : [int],
  "RefTurn" : [int],
  "Auth" : {
    "AuthCode" : [string],
    "ClientName" : [string],
    "SequenceNumber" : [int],
    "SessionId" : [int],
    "TeamName" : [string]
  }
}
```

Request example

```
{
  "PlayerId" : 2,
  "RefTurn" : 0,
  "Auth" : {
    "AuthCode" : "1efb12da2fdc6edbe43a34493352811e8fd59a5d",
    "ClientName" : "Player2",
    "SequenceNumber" : 6,
    "SessionId" : 325487283,
    "TeamName" : "Auth"
  }
}
```

Response

```
{
  "FinishComment" : [string],
  "FinishCondition" : [string],
  "GameFinished" : [bool],
  "TurnComplete" : [bool],
  "YourTurn" : [bool],
  "Message" : [string],
  "Status" : [string]
}
```

Response examples

```
{
  "FinishComment" : null,
  "FinishCondition" : "Play",
  "GameFinished" : false,
  "TurnComplete" : false,
  "YourTurn" : true,
  "Message" : null,
  "Status" : "OK"
}

{
  "FinishComment" : null,
  "FinishCondition" : "Won",
  "GameFinished" : true,
  "TurnComplete" : true,
  "YourTurn" : true,
  "Message" : null,
  "Status" : "OK"
}
```

Method	PerformMove
Call link	/ClientService.svc/json/PerformMove
HTTP method	POST
Request	`{ "PlayerId" : [int], "Turn" : { "Col" : [int], "Row" : [int] }, "Pass" : [bool], "Auth" : { "AuthCode" : [string], "ClientName" : [string], "SequenceNumber" : [int], "SessionId" : [int], "TeamName" : [string] } }`
Request example	`{ "PlayerId" : 1, "Turn" : { "Col" : 3, "Row" : 5 }, "Pass" : false, "Auth" : { "AuthCode" : "ad096fffcfaca914352d5d7934b4cf3e1bc40fe5", "ClientName" : "Player1", "SequenceNumber" : 14, "SessionId" : 1724163314, "TeamName" : "Auth" } }`
Response	`{ "Message" : [string], "Status" : [string] }`
Response examples	`{ "Message" : null, "Status" : "OK" }`

Method	GetPlayerView
Call link	/ClientService.svc/json/GetPlayerView
HTTP method	POST

Request	`{`
	  `"PlayerId" : [int],`
	  `"Auth" : {`
	    `"AuthCode" : [string],`
	    `"ClientName" : [string],`
	    `"SequenceNumber" : [int],`
	    `"SessionId" : [int],`
	    `"TeamName" : [string]`
	  `}`
	`}`

Request example	`{`
	  `"PlayerId" : 1,`
	  `"Auth" : {`
	    `"AuthCode" : "e1d20b9fc301fd39f30bd3c15522b79316668c4a",`
	    `"ClientName" : "Player1",`
	    `"SequenceNumber" : 24,`
	    `"SessionId" : 1724163314,`
	    `"TeamName" : "Auth"`
	  `}`
	`}`

Response	`{`
	  `"GameState" : [string],`
	  `"Index" : [int],`
	  `"Map" : {`
	    `"Height":[int],`
	    `"Rows" : [string[]],`
	    `"Width":[int]`
	  `},`
	  `"PlayerStates" : [playerState[]],`
	  `"Turn" : [int],`
	  `"Message" : [string],`
	  `"Status" : [string]`
	`}`
	
	playerState:
	`{`
	  `"Comment" : [string],`
	  `"Condition" : [string]`
	`}`

Response examples	`{`
	  `"GameState" : "Play",`
	  `"Index" : 0,`

```
  "Map" : {
    "Height" : 10,
    "Rows" : [
      "**********",
      "**********",
      "**********",
      "*****0****",
      "****00****",
      "****10****",
      "**********",
      "**********",
      "**********",
      "**********"
    ],
    "Width" : 10
  },
"PlayerStates" : [
    {
      "Comment" : null,
      "Condition" : "Play",
    },
    {
      "Comment" : null,
      "Condition" : "Play",
    }
  ],
  "Turn" : 2,
  "Message" : null,
  "Status" : "OK"
}
```

Method	LeaveGame
Call link	/ClientService.svc/json/LeaveGame
HTTP method	POST
Request	`{ "PlayerId" : [int], "Auth" : { "AuthCode" : [string], "ClientName" : [string], "SequenceNumber" : [int], "SessionId" : [int], "TeamName" : [string] } }`
Request example	`{ "PlayerId" : 2, "Auth" : { "AuthCode" : "53471b3bc3653e88db05c4f2b5886cd499430e9d", "ClientName" : "Player2", "SequenceNumber" : 17, "SessionId" : 142366330, "TeamName" : "Auth" } }`
Response	`{ "Message" : [string], "Status" : [string] }`
Response examples	`{ "Message" : null, "Status" : "OK" }` `{ "Message" : "System.ApplicationException: Player is not in a game", "Status" : "FAIL" }`