Channel Python API Overview

- The Channel API creates a persistent connection between your application and Google servers, allowing your application to send messages to JavaScript clients in real time without the use of polling.
- This is useful for applications designed to update users about new information immediately.
- Some example use-cases include collaborative applications, multi-player games, or chat rooms.
- In general, using the Channel API is a better choice than polling in situations where updates can't be predicted or scripted, such as when relaying information between human users or from events not generated systematically.

Life of a typical channel message

- These two diagrams illustrate the life of a typical example message sent via Channel API between two different clients using one possible implementation of Channel API.

- It shows the JavaScript client explicitly requests a token and sends its Client ID to the server.

- In contrast, you could choose to design your application to inject the token into the client before the page loads in the browser, or some other implementation if preferred.

Javascript Client

- With a unique token channel to connect channel, establish a long connection with the sever.
- Monitor channel data and updates to the user.
- Send data to sever.

Server

- Server: is the GAE server, which is responsible for: Each Javascript client to create a unique channel.

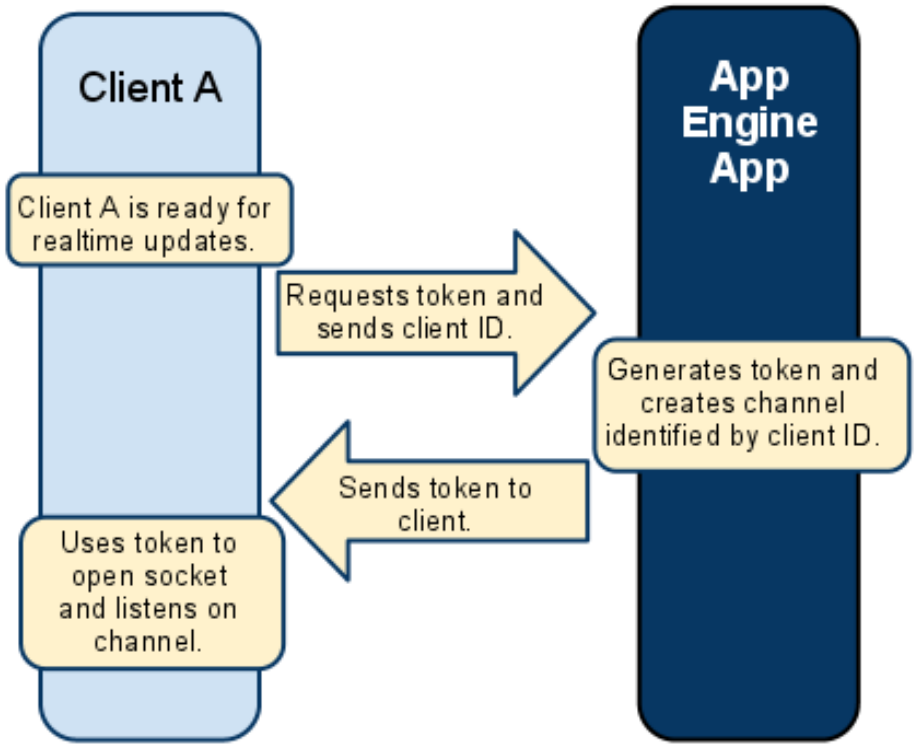
1. Create and send a unique token to the client.

2. POST client receives data transmitted.

3. To send data to the client through the channel.

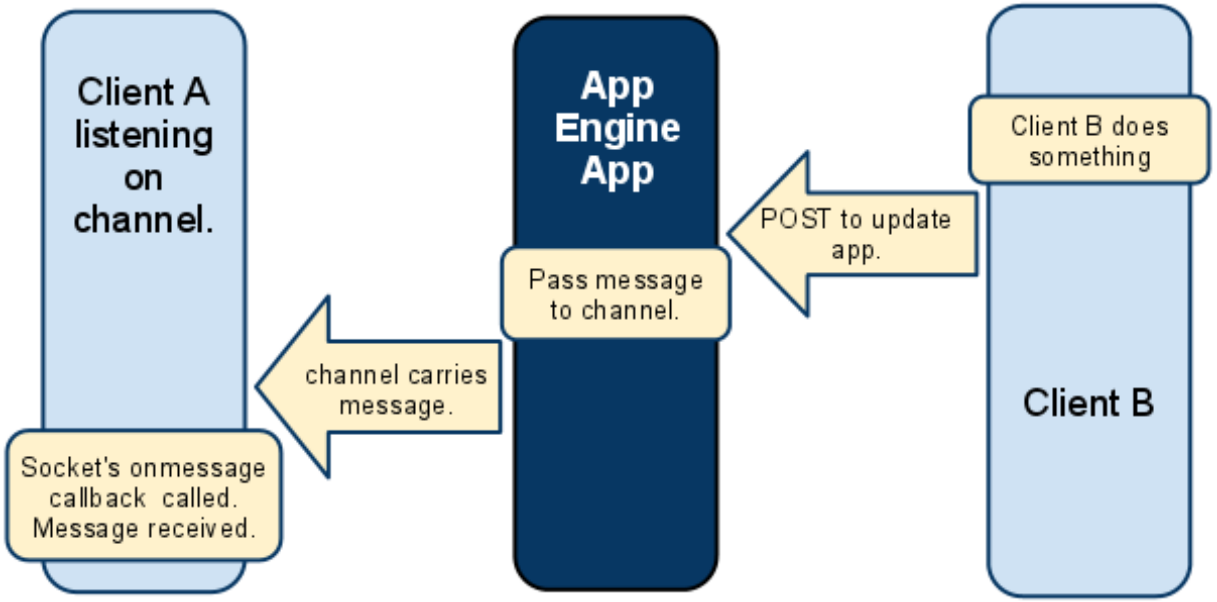
Life of a typical channel message

- The server uses Client A's Client ID to create a channel and then sends the token for that channel back to Client A.
- Client A uses the token to open a socket and listen for updates on the channel.

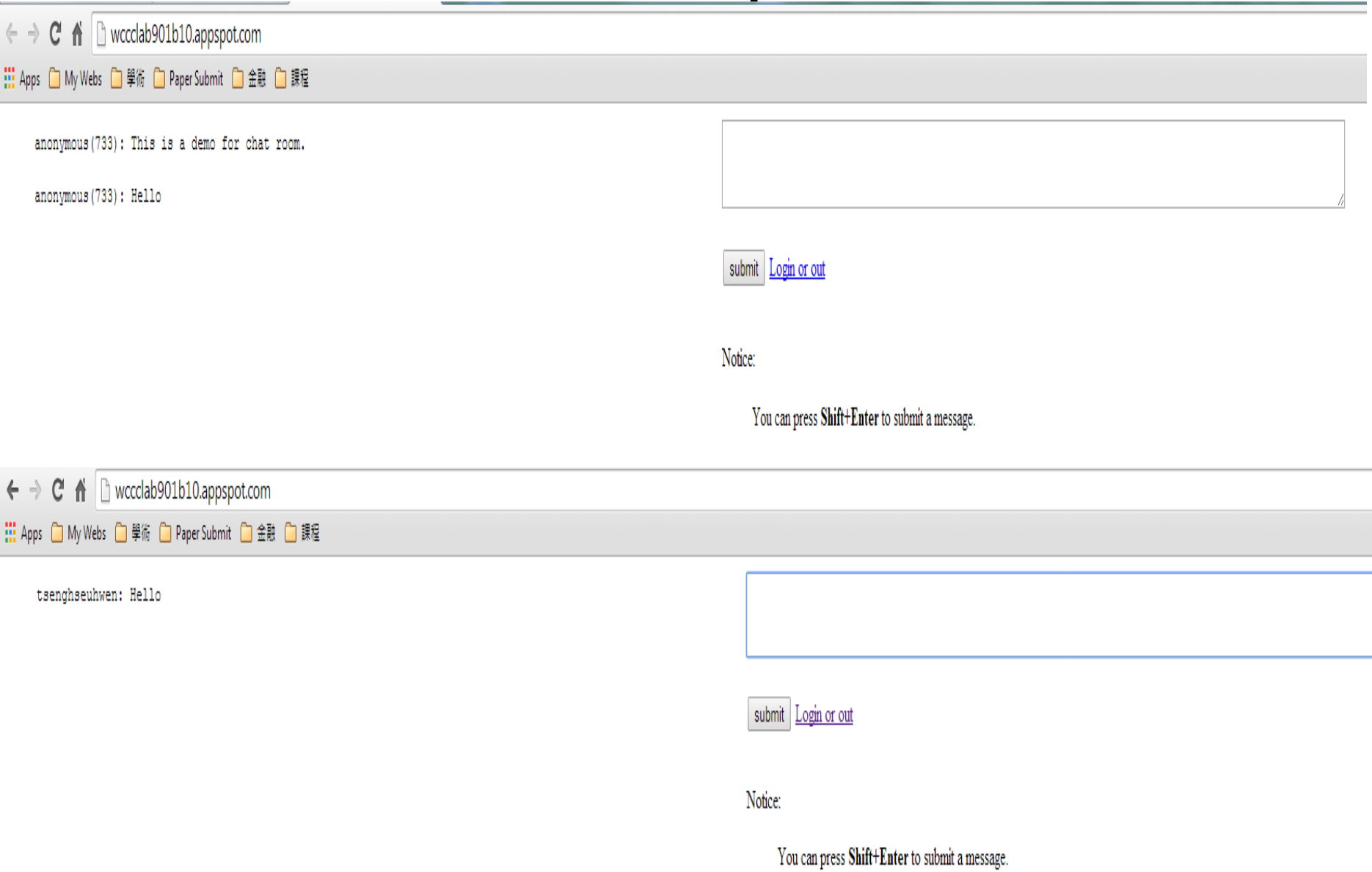


Life of a typical channel message

- This diagram shows Client B sending a message using POST to the server. The server processes the message and sends it to Client A over the channel.
- Client A receives the message and makes use of the new information.



Example


wccclab901b10.appspot.com
Apps
My Webs
學術
Paper Submit
金融
課程
anonymous(733): This is a demo for chat room.
anonymous(733): Hello
submit
Login or out
Notice:
You can press Shift+Enter to submit a message.
wccclab901b10.appspot.com
Apps
My Webs
學術
Paper Submit
金融
課程
tsenghseuhwen: Hello
submit
Login or out
Notice:
You can press Shift+Enter to submit a message.

Example

```
Class  GetTokenHandler (webapp.RequestHandler) :
    def  get (self) :
        user = users.get_current_user () if user:
            CHANNEL_ID = ID = user.email () else :
            ID = random.randint ( 1 , 1000 )
            CHANNEL_ID = 'Anonymous ( % s) ' % ID
        token = channel.create_channel (CHANNEL_ID)
        tokens = memcache.get ( 'tokens' ) or {}
        tokens [token] = ID
        memcache.set ( 'tokens' , tokens)
        self.response.out.write (token)
```

- If the user is not logged in, then it generates a random number as channel_id.
- Note channel_id and token can not be discarded, because later we will use, so we put memcache .
- http://wccclab901b10.appspot.com/

Chat Rooms

```
var token;
 Function  get_token () {
    $ get (. '/ get_token' , Function (Data) {
        token = Data;
        OpenChannel ();
    });
}
```

- After receiving the token, you can use it to connect the Channel.

```
Function  OpenChannel () {
    var channel = new goog.appengine.Channel (token);
     var handler = {
       'onopen' : onOpen,
       'onMessage' : onMessage,
       'onerror' : Function () { },
       'OnClose' : Function ( ) { }
    };
    channel.open (handler);
}
```

Chat Rooms

1. With new goog.appengine.Channel (token) to create a Channel object. (Note: need to load / _ah / channel / jsapi This JavaScript file to use this class.)

2. Construct a handler to process the information Channel sent.

3. The handler as a parameter, call the open method of Channel objects to create socket connections.

Four Kinds of Events

1. onopen: Called when the socket is successfully established. It does not matter whether the definition, let me tell the server where new users join the chat room:

```
function onOpen() {
    $.post('/open', {'token': token});
}
```

```
class OpenHandler(webapp.RequestHandler):
    def post(self):
        token = self.request.get('token')
        if not token:
            return
        tokens = memcache.get('tokens')
        if tokens:
            id = tokens.get(token, '')
            if id:
                if isinstance(id, int):
                    user_name = u'天朝匿名用户(%s)' % id
                else:
                    user_name = id.split('@')[0]
                message = user_name + u'加入了聊天室'
                message = simplejson.dumps(message)
                send_to_all(message, tokens)
```

Four Kinds of Events

- The client sends token, in the memcache server where to get Client ID based on token, and use send_to_all () to notify other Client broadcasts a new user added.

```
def  send_to_all (message, tokens = None) :
    if  Not tokens:
        tokens = memcache.get ( 'tokens' )
     if tokens:
         for token, ID in tokens.iteritems ():
             if isinstance (ID, int):
                ID = 'Anonymous (% s) ' % ID
            channel.send_message (ID, message)
```

- Note that even without the user receives Channel, channel.send_message () method does not throw an exception, so it is necessary to clear the channel themselves no longer use.

Four Kinds of Events

- onmessage: Called when a message is received Channel sent. You do not realize it does not make sense, and here I use it to display the message sent by the user.

```
Function  onMessage (m) {
    var message = $ parseJSON (m.data);.
    message = message.replace ( / & / g , '&' .) replace ( / </ g , '<' )
replace (. /> / g , '>' );
    $ msg.prepend ( '<blockquote> <pre>' + message + '</ pre> </ blockquote>'
);
}
```

```
Class  ReceiveHandler (webapp.RequestHandler) :
    def  Post (self) :
        token = self.request.get ( 'token' )
         if  Not token:
             return
        message = self.request.get ( 'content' )
         if  Not message:
             return
        tokens = memcache.get ( 'tokens' )
         if tokens:
            ID = tokens.get (token, '' )
             if ID:
                 if isinstance (ID, int):
                    user_name = u 'heavenly anonymous user (% s)' % ID
                 else :
                    user_name = id.split ( '@' ) [ 0 ]
                message = '% s:% s' % (user_name, message)
                message = simplejson.dumps (message) if len (message)>
channel.MAXIMUM_MESSAGE_LENGTH:
                     return
                send_to_all (message)
```

Four Kinds of Events

- onerror: Called when the socket error.

- onclose: Called when the Channel off.
  - Note Channel after the establishment of two hours expire, then calls onclose and onerror, can capture this event, access token again to connect Channel.

Client POST

```
Function submit () {
    . $ Ajax ({
        url: '/ post_msg' ,
        type: 'POST' ,
        Data: { 'token' : token, 'content' : $ content.val ()}
    });
    $ content.val ( '' .) Focus ();
}

$ content.keypress (Function (e) { if (e.shiftKey && e.keyCode == 13 ) {
        submit (); return false;
    }
});
$ ( '# submit_msg ' .) click (submit);
```

- This code calls the service side of ReceiveHandler.post (), thus associating with onmessage.

Release Token

```
. $ (Window) bind ( 'beforeunload' , Function () {
    $ Post (. '/ del_token' , { 'token' : token});
})
```

```
Class  ReleaseTokenHandler (webapp.RequestHandler) :
    def  Post (self) :
        token = self.request.get ( 'token' )
         if  Not token:
             return
        tokens = memcache.get ( 'tokens' )
         if tokens:
            ID = tokens.get ( token, '' )
             if ID:
                 if isinstance (ID, int):
                    user_name = u 'heavenly anonymous user (% s)' % ID
                 else :
                    user_name = id.split ( '@' ) [ 0 ]
                message = user_name + u 'left the chat room'
                message = simplejson.dumps (message) del tokens [token]
                memcache.set ( 'tokens' , tokens)
                send_to_all (message, tokens)
```

```
Class  LoginOrOut (webapp.RequestHandler) :
    def  get (self) :
        if users.get_current_user ():
            self.redirect (users.create_logout_url ( '/' ))
         else :
            self.redirect (users.create_login_url ( '/' ))
```

Tic Tac Toe Game (Console)

- We uses two **for loops** to go through a list variable called map.
- This variable is a two dimensional **list** which will hold the info about what's in each position.

```
def print_board():
    for i in range(0,3):
        for j in range(0,3):
            print map[2-i][j],
            if j != 2:
                print "|",
        print ""

turn = "X"
map = [[" "," "," "],
       [" "," "," "],
       [" "," "," "]]
done = False
```

```
X 's turn

Please select position by typing in a number between 1 and 9, see below for whic
h number that is which position...
7|8|9
4|5|6
1|2|3

Select: 5
  |   |
  | X |
  |   |
O 's turn

Please select position by typing in a number between 1 and 9, see below for whic
h number that is which position...
7|8|9
4|5|6
1|2|3

Select: 4
  |   |
O | X |
  |   |
X 's turn

Please select position by typing in a number between 1 and 9, see below for whic
h number that is which position...
7|8|9
4|5|6
1|2|3
```

Tic Tac Toe Game (Console)

1. We check if all 3 squares in all **horizontal** and **vertical** lines are the same and not " ".
   - This is so it won't think an completely empty line is a line with 3 in a row.
2. Then it checks the two **diagonally** lines in the same way.
3. If at least one of these 8 lines are a winning line we will print out turn, "won!!!" and also return the value True. the turn variable will hold which player who's in turn so the message will be either "X won!!!" or "O won!!!".

```
def check_done():
    for i in range(0,3):
        if map[i][0] == map[i][1] == map[i][2] != " " \
        or map[0][i] == map[1][i] == map[2][i] != " ":
            print turn, "won!!!"
            return True

    if map[0][0] == map[1][1] == map[2][2] != " " \
    or map[0][2] == map[1][1] == map[2][0] != " ":
        print turn, "won!!!"
        return True

    if " " not in map[0] and " " not in map[1] and " " not in map[2]:
        print "Draw"
        return True


    return False
```

Tic Tac Toe Game (Console)

- We store the current users "name" at the right position (with the X and Y values), set move to True, check if we're done and stores that in done.
- If the game isn't over, change who's next to move and then we have two lines to print an error message if the try block failed in some way.

```
map[Y][X] = turn
                    moved = True
                    done = check_done()

                    if done == False:
                        if turn == "X":
                            turn = "O"
                        else:
                            turn = "X"


        except:
            print "You need to add a numeric value"
```

Tic Tac Toe Game (Console)

```
while done != True:
    print_board()

    print turn, "'s turn"
    print

    moved = False
    while moved != True:
        print "Please select position by typing in a number between 1 and 9, see below for which number that is which position..."
        print "7|8|9"
        print "4|5|6"
        print "1|2|3"
        print

        try:
            pos = input("Select: ")
            if pos <=9 and pos >=1:
                Y = pos/3
                X = pos%3
                if X != 0:
                    X -=1
                else:
                     X = 2
                     Y -=1

                if map[Y][X] == " ":
                    map[Y][X] = turn
                    moved = True
                    done = check_done()

                    if done == False:
                        if turn == "X":
                            turn = "O"
                        else:
                            turn = "X"


        except:
            print "You need to add a numeric value"
```

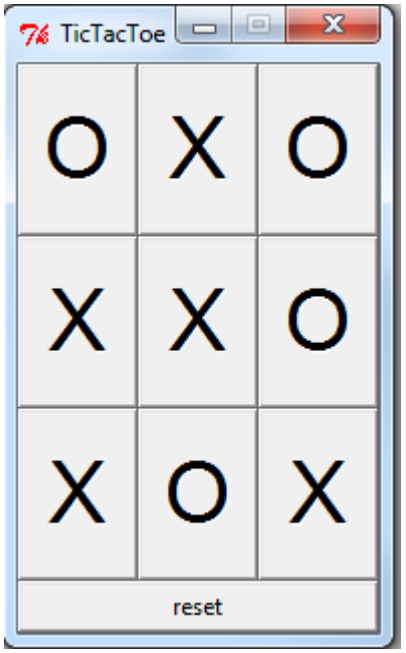
Tic Tac Toe Game (GUI)

```
from Tkinter import Tk, Button
from tkFont import Font
from copy import deepcopy

class Board:

  def __init__(self,other=None):
    self.player = 'X'
    self.opponent = 'O'
    self.empty = '.'
    self.size = 3
    self.fields = {}
    for y in range(self.size):
      for x in range(self.size):
        self.fields[x,y] = self.empty
    # copy constructor
    if other:
      self.__dict__ = deepcopy(other.__dict__)

  def move(self,x,y):
    board = Board(self)
    board.fields[x,y] = board.player
    (board.player,board.opponent) = (board.opponent,board.player)
    return board
```



Tic Tac Toe Game (GUI)

```
def __minimax(self, player):
  if self.won():
    if player:
      return (-1,None)
    else:
      return (+1,None)
  elif self.tied():
    return (0,None)
  elif player:
    best = (-2,None)
    for x,y in self.fields:
      if self.fields[x,y]==self.empty:
        value = self.move(x,y).__minimax(not player)[0]
        if value>best[0]:
          best = (value,(x,y))
    return best
  else:
    best = (+2,None)
    for x,y in self.fields:
      if self.fields[x,y]==self.empty:
        value = self.move(x,y).__minimax(not player)[0]
        if value<best[0]:
          best = (value,(x,y))
    return best
```

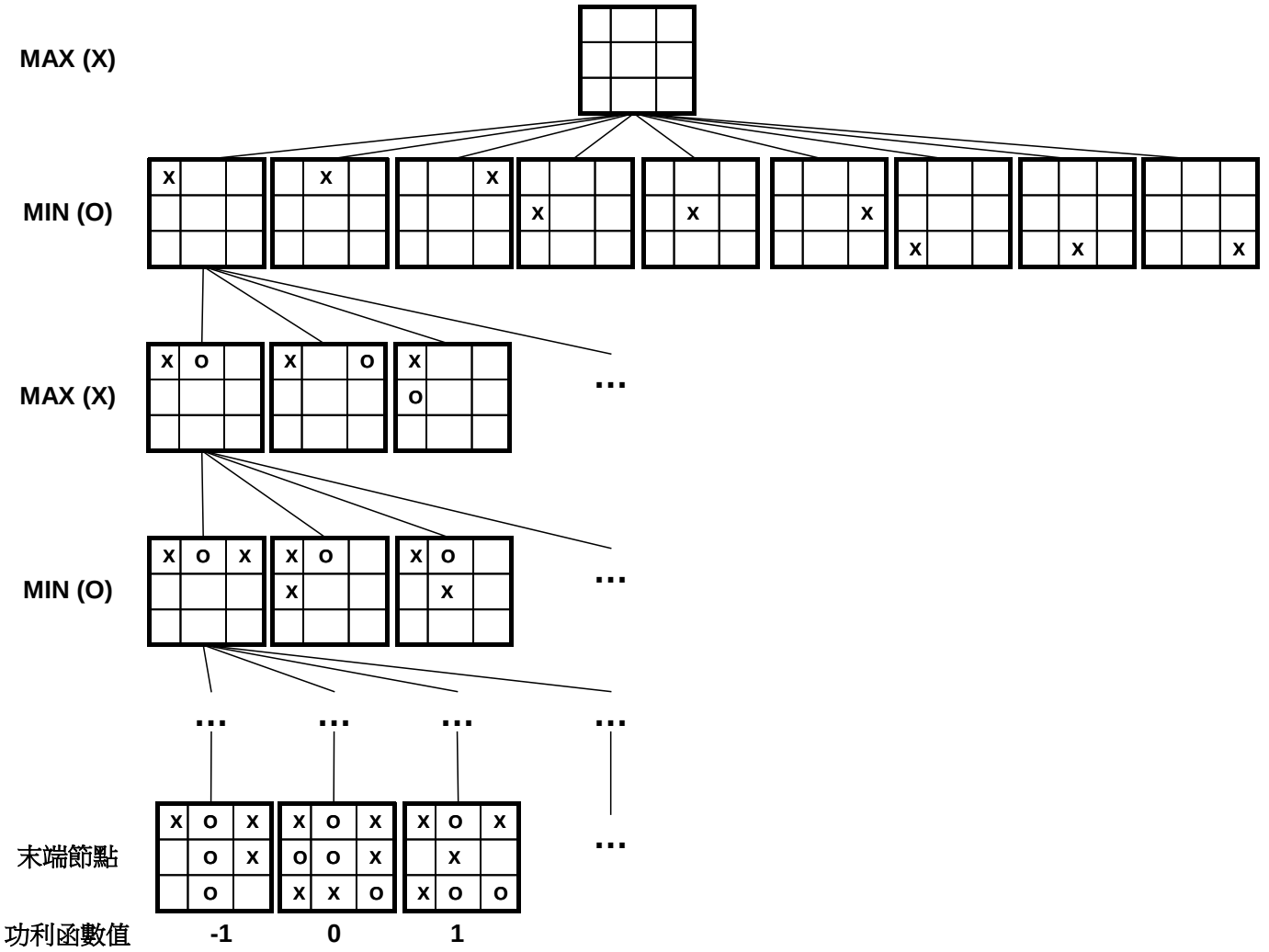
最小值最大化的遊戲策略 (Minimax Game Decision)

- 假設一個遊戲中有兩位參與者，為方便起見我們稱他們為MAX和MIN。
- 由MAX先移動，且兩人輪流移動直到遊戲結束。
- 贏的人會得到獎賞(或輸的一方得到處罰)。

一個遊戲若具備以下條件，則可以被轉換成為一個搜尋的問題：

- 初始狀態(initial state)，包含初始位置及哪一方先移動等。
- 運算元集合(a set of operators)，指一個玩家在遊戲中可作的動作。
- 結束測試(terminal test)，決定何時遊戲會結束。遊戲結束的地方稱為結束狀態(terminal states)。
- 功利函數(utility function)，或稱回報函數(payoff function)，對於遊戲的結果給予一數值。

井字遊戲的搜尋樹(search tree)



Tic Tac Toe Game (GUI)

```
def won(self):
  # horizontal
  for y in range(self.size):
    winning = []
    for x in range(self.size):
      if self.fields[x,y] == self.opponent:
        winning.append((x,y))
    if len(winning) == self.size:
      return winning
  # vertical
  for x in range(self.size):
    winning = []
    for y in range(self.size):
      if self.fields[x,y] == self.opponent:
        winning.append((x,y))
    if len(winning) == self.size:
      return winning
  # diagonal
  winning = []
  for y in range(self.size):
    x = y
    if self.fields[x,y] == self.opponent:
      winning.append((x,y))
  if len(winning) == self.size:
    return winning
  # other diagonal
  winning = []
  for y in range(self.size):
    x = self.size-1-y
    if self.fields[x,y] == self.opponent:
      winning.append((x,y))
  if len(winning) == self.size:
    return winning
  # default
  return None
```

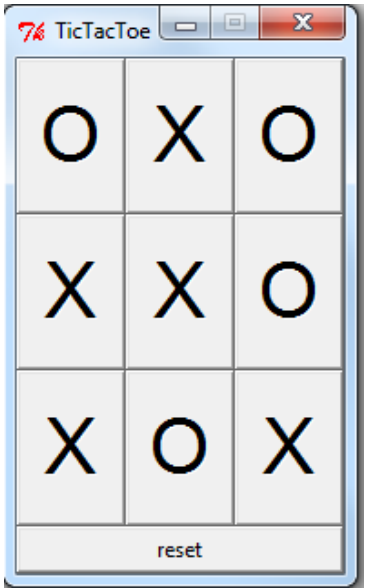
Tic Tac Toe Game (GUI)

```
class GUI:

  def __init__(self):
    self.app = Tk()
    self.app.title('TicTacToe')
    self.app.resizable(width=False, height=False)
    self.board = Board()
    self.font = Font(family="Helvetica", size=32)
    self.buttons = {}
    for x,y in self.board.fields:
      handler = lambda x=x,y=y: self.move(x,y)
      button = Button(self.app, command=handler, font=self.font, width=2, height=1)
      button.grid(row=y, column=x)
      self.buttons[x,y] = button
    handler = lambda: self.reset()
    button = Button(self.app, text='reset', command=handler)
    button.grid(row=self.board.size+1, column=0, columnspan=self.board.size, sticky="WE")
    self.update()

  def reset(self):
    self.board = Board()
    self.update()

  def move(self,x,y):
    self.app.config(cursor="watch")
    self.app.update()
    self.board = self.board.move(x,y)
    self.update()
    move = self.board.best()
    if move:
      self.board = self.board.move(*move)
      self.update()
    self.app.config(cursor="")
```



Tic Tac Toe Game (GUI)

```
def update(self):
  for (x,y) in self.board.fields:
    text = self.board.fields[x,y]
    self.buttons[x,y]['text'] = text
    self.buttons[x,y]['disabledforeground'] = 'black'
    if text==self.board.empty:
      self.buttons[x,y]['state'] = 'normal'
    else:
      self.buttons[x,y]['state'] = 'disabled'
  winning = self.board.won()
  if winning:
    for x,y in winning:
      self.buttons[x,y]['disabledforeground'] = 'red'
    for x,y in self.buttons:
      self.buttons[x,y]['state'] = 'disabled'
  for (x,y) in self.board.fields:
    self.buttons[x,y].update()

def mainloop(self):
  self.app.mainloop()
```

Example Tic Tac Toe application

Creating and connecting to a channel

- When a user visits the Tic Tac Toe game for the first time, two things happen:
  1. The game server injects a token into the html page sent to the client. The client uses this token to open a socket and listen for updates on the channel.
  2. The game server provides the user with a URL they can share with a friend in order to invite him or her to join the game.
- To initiate the process of creating a channel, JavaScript client pages need to call the create_channel() method and get a token that the client page can use to connect to the channel.
- When calling create_channel(), they should use a key that the application can use to uniquely identify the client.

```
import jinja2
import os
import webapp2
from google.appengine.api import channel
from google.appengine.api import users

class MainPage(webapp2.RequestHandler):
  """This page is responsible for showing the game UI. It may also
  create a new game or add the currently-logged in user to a game."""

  def get(self):
    user = users.get_current_user()
    if not user:
      self.redirect(users.create_login_url(self.request.uri))
      return

    game_key = self.request.get('gamekey')
    game = None
    if not game_key:
      # If no game was specified, create a new game and make this user
      # the 'X' player.
      game_key = user.user_id()
      game = Game(key_name = game_key,
                  userX = user,
                  moveX = True,
                  board = '         ')
      game.put()
    else:
      game = Game.get_by_key_name(game_key)
      if not game.userO and game.userX != user:
        # If this game has no 'O' player, then make the current user
        # the 'O' player.
        game.userO = user
        game.put()
```

Example

```
    else:
      game = Game.get_by_key_name(game_key)
      if not game.userO and game.userX != user:
        # If this game has no 'O' player, then make the current user
        # the 'O' player.
        game.userO = user
        game.put()

    token = channel.create_channel(user.user_id() + game_key)
    template_values = {'token': token,
                       'me': user.user_id(),
                       'game_key': game_key
                       }
    template = jinja_environment.get_template('index.html')
    self.response.out.write(template.render(template_values))

jinja_environment = jinja2.Environment(
    loader=jinja2.FileSystemLoader(os.path.dirname(__file__)))
app = webapp2.WSGIApplication([('/', MainPage)],
                              debug=True)
```

Client Example

- The client creates a new goog.appengine.Channel object using the token provided by the server.

```
<body>
  <script>
    channel = new goog.appengine.Channel('{{ token }}');
    socket = channel.open();
    socket.onopen = onOpened;
    socket.onmessage = onMessage;
    socket.onerror = onError;
    socket.onclose = onClose;
  </script>
</body>
```

- The game client uses the Channel object's open() method to create a socket.
- The client also sets callback functions on the socket to be called when the state of the socket changes.

Opening the socket

- When the Tic Tac Toe client is ready to receive messages, it calls the onOpened() function, which it set to the socket's onopen callback.
- The onOpened function also updates the UI for the user to indicate that the game is ready to play and sends a POST message to the server to ask it to send the latest game state.
- The following client-side JavaScript code implements this functionality:

```
sendMessage = function(path, opt_param) {
  path += '?g=' + state.game_key;
  if (opt_param) {
    path += '&' + opt_param;
  }
  var xhr = new XMLHttpRequest();
  xhr.open('POST', path, true);
  xhr.send();
};

onOpened = function() {
  connected = true;
  sendMessage('opened');
  updateBoard();
};
```

Note that the application defines sendMessage() as a wrapper around XmlHttpRequest, which the client uses to send messages to the server.

Updating the Game State

- The Tic Tac Toe Javascript client uses an onClick handler called moveInSquare to handle mouse clicks in the board.
- When a player makes a move in our Tic Tac Toe game by clicking on a square, the client uses XmlHttpRequest to send a POST message to the application with the proposed move.

```
moveInSquare = function(id) {
  if (isMyMove() && state.board[id] == ' ') {
    sendMessage('/move', 'i=' + id);
  }
}
```

Validating and sending the new game state

- Use an HTTP request to send messages from the client to the server.
  - In this example, when the server receives the client's message via an HTTP request, it first uses its request handler to validate the move.

- Then, if the move is legal, the server uses the channel.send_message() method to send messages indicating the new state of the board to both clients.

- The MovePage RequestHandler is called in response to the client's POST in the sendMessage call above.

Validating and sending the new game state

- This handler is responsible for validating the move and broadcasting the new board state to the clients.

```
class MovePage(webapp2.RequestHandler):

  def post(self):
    game = GameFromRequest(self.request).get_game()
    user = users.get_current_user()
    if game and user:
      id = int(self.request.get('i'))
      GameUpdater(game).make_move(id, user)
```

- The GameFromRequest class uses the gamekey parameter on the POST to retrieve the current game.

```
class GameFromRequest():
  game = None;

  def __init__(self, request):
    user = users.get_current_user()
    game_key = request.get('gamekey')
    if user and game_key:
      self.game = Game.get_by_key_name(game_key)

  def get_game(self):
    return self.game
```

```
class GameUpdater():
  """Creates an object to store the game's state, and handles validating moves
  and broadcasting updates to the game."""
  game = None

  def __init__(self, game):
    self.game = game

  def get_game_message(self):
    # The gameUpdate object is sent to the client to render the state of a game.
    gameUpdate = {
      'board': self.game.board,
      'userX': self.game.userX.user_id(),
      'userO': '' if not self.game.userO else self.game.userO.user_id(),
      'moveX': self.game.moveX,
      'winner': self.game.winner,
      'winningBoard': self.game.winning_board
    }
    return simplejson.dumps(gameUpdate)

  def send_update(self):
    message = self.get_game_message()
    channel.send_message(self.game.userX.user_id() + self.game.key().name(),
message)
    if self.game.userO:
      channel.send_message(self.game.userO.user_id() + self.game.key().name(),
message)

  def check_win(self):
    if self.game.moveX:
      # O just moved, check for O wins
      wins = Wins().o_wins
      potential_winner = self.game.userO.user_id()
    else:
      # X just moved, check for X wins
      wins = Wins().x_wins
      potential_winner = self.game.userX.user_id()
```

ove is valid and, if it
l to send updates
ate.

Example

```
    for win in wins:
      if win.match(self.game.board):
        self.game.winner = potential_winner
        self.game.winning_board = win.pattern
        return

  def make_move(self, position, user):
    if position >= 0 and user == self.game.userX or user == self.game.userO:
      if self.game.moveX == (user == self.game.userX):
        boardList = list(self.game.board)
        if (boardList[position] == ' '):
          boardList[position] = 'X' if self.game.moveX else 'O'
          self.game.board = "".join(boardList)
          self.game.moveX = not self.game.moveX
          self.check_win()
          self.game.put()
          self.send_update()
          return
```

Tracking client connections and disconnections

- Applications may register to be notified when a client connects to or disconnects from a channel.
- You can enable this inbound service in app.yaml:

```
inbound_services:
- channel_presence
```

- When you enable channel_presence, your application receives POSTs to the following URL paths:
- POSTs to /_ah/channel/connected/ signal that the client has connected to the channel and can receive messages.
- POSTs to /_ah/channel/disconnected/ signal that the client has disconnected from the channel.

Tracking client connections and disconnections

- Your application can register handlers to these paths in order to receive notifications. You can use these notifications to track which clients are currently connected.
- The "from" parameter in the POST identifies the client_id used to create the channel whose presence has changed.

```
# In the handler for _ah/channel/connected/
client_id = self.request.get('from')
```

```
    while 1:
        # Get the list sockets which are ready to be read through select
        read_sockets,write_sockets,error_sockets = select.select(CONNECTION_LIST,[],

        for sock in read_sockets:
            #New connection
            if sock == server_socket:
                # Handle the case in which there is a new connection recieved through
                sockfd, addr = server_socket.accept()
                CONNECTION_LIST.append(sockfd)
                print "Client (%s, %s) connected" % addr

                broadcast_data(sockfd, "[%s:%s] entered room\n" % addr)

            #Some incoming message from a client
            else:
                # Data recieved from client, process it
                try:
                    #In Windows, sometimes when a TCP program closes abruptly,
                    # a "Connection reset by peer" exception will be thrown
                    data = sock.recv(RECV_BUFFER)
                    if data:
                        broadcast_data(sock, "\r" + '<' + str(sock.getpeername()) +

                except:
                    broadcast_data(sock, "Client (%s, %s) is offline" % addr)
                    print "Client (%s, %s) is offline" % addr
                    sock.close()
                    CONNECTION_LIST.remove(sock)
                    continue

    server_socket.close()
```

```
$ python chat_server.py
Chat server started on port 5000
```

Python Interpreters

- http://www.python.org/download/
- http://pyaiml.sourceforge.net/
- http://www.py2exe.org/
- http://www.activestate.com/Products/activepython/
- http://www.wingware.com/
- http://pythonide.blogspot.com/
- Many more…

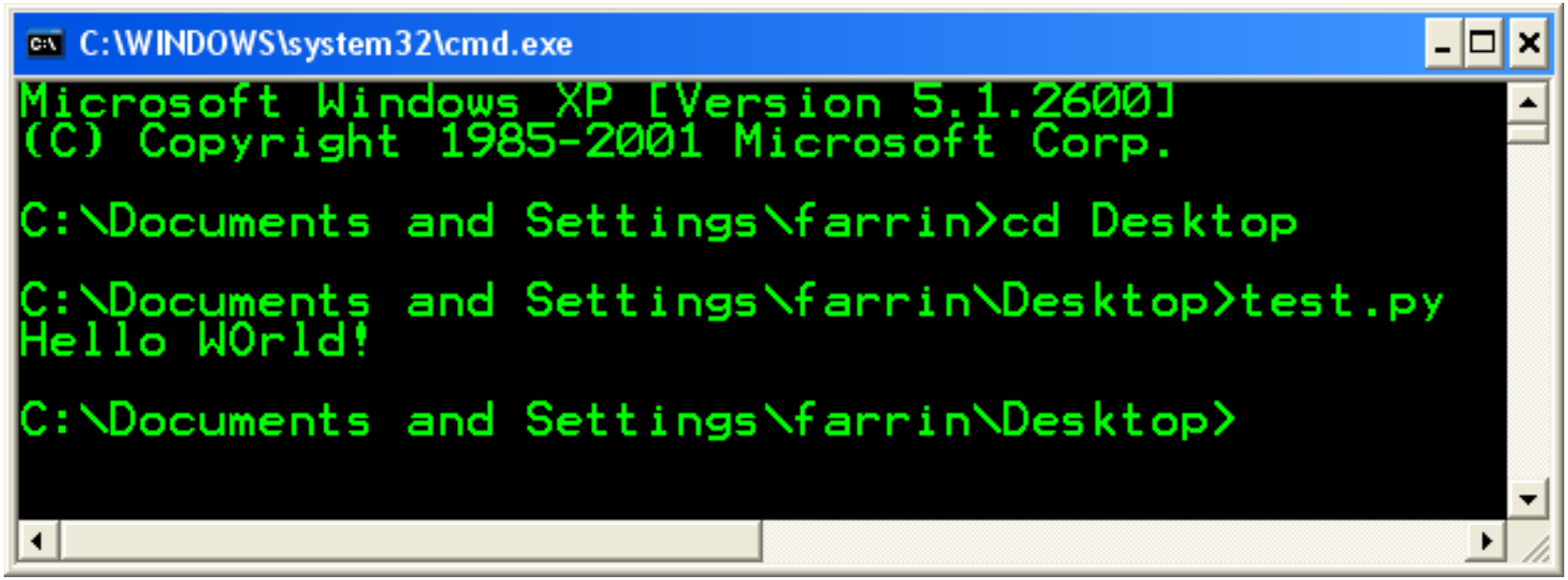
Python on your systems

- Its easy! Go to http://www.python.org/download/
- Download your architecture binary, or source
- Install, make, build whatever you need to do… plenty of info on installation in readmes
- Make your first program! (a simple on like the hello world one will do just fine)
- Two ways of running python code. Either in an interpreter or in a file ran as an executable

Running Python

- Windows XP – double click the icon or call it 

from the command line as such:



Python Interpreter

Python (command line)

```
Python 2.5.2 (r252:60911, Feb 21 2008, 13:11:45) [MSC v.1310 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'hello world'
hello world
>>> x = 'roses'
>>> y = 12
>>> Roy_G_Biv = [('red', 'orange','yellow'),'green',('blue',['indigo','violet'])
]
>>> MyAwesomeVar = [x, y, Roy_G_Biv]
>>> print MyAwesomeVar
['roses', 12, [('red', 'orange', 'yellow'), 'green', ('blue', ['indigo', 'violet
'])]]
>>> print MyAwesomeVar[0]+' are '+MyAwesomeVar[2:3][0][0][0]
roses are red
>>> print MyAwesomeVar[2:3][0][2][1][1]+'s are '+MyAwesomeVar[2:3][0][2][0]
violets are blue
>>> print str(MyAwesomeVar[1])+' '+MyAwesomeVar[0]+' for my love.'
12 roses for my love.
>>> dict = {'place': 'mantle','where': 'above', 'myLove': True}
>>> if(dict["myLove"]): print 'the ' +dict["place"]+' I put them '+dict["where"]


...
the mantle I put them above
>>>
```

Python for the future

- Python 3.0
  - Will not be Backwards compatible, they are attempting to fix “perceived” security flaws.
  - Print statement will become a print function.
  - All text strings will be unicode.
  - Support of optional function annotation, that can be used for informal type declarations and other purposes.

Bibliography

- http://it.metr.ou.edu/byteofpython/features-of-python.html
- http://codesyntax.netfirms.com/lang-python.htm
- http://www.python.org/
- Sebesta, Robert W., Concepts of Programming Languages: 8th ed. 2007
- http://www.python.org/~guido/