Lab 3 Solutions

Client: GameEnv

```
#
# PURPOSE:
# A simple container for storing the values 'num_rows', 'num_cols',
# 'player_id', and 'grid'
#
# PARAMETERS:
# num_rows: The number of rows in the grid
# num_cols: The number of columns in the grid
# player_id: The ID of the player (a string, which will be displayed
# in the grid)
# grid: The game board (a 2D array of characters, representing
# players, treasures, and blank spaces)
#
# RETURN/SIDE EFFECTS:
# N/A
#
# NOTES:
# N/A
#
class Game_Env:
    def __init__(self, num_rows, num_cols, player_id, grid):
        self.num_rows = num_rows
        self.num_cols = num_cols
        self.player_id = player_id
        self.grid = grid
```

Client: displayGrid

```
#
# PURPOSE:
# Displays the grid contained in 'game_env.grid' on the screen
#
# PARAMETERS:
# stdscr: Must be a reference to a valid curses screen
# game_env: Contains a valid game environment
#
# RETURN/SIDE EFFECTS:
# N/A
#
# NOTES:
# N/A
#
def displayGrid(stdscr, game_env):
    stdscr.clear()
    pos = 0
    for row in range(game_env.num_rows):
        for col in range(game_env.num_cols):
            stdscr.addstr(row, col * SPACER, game_env.grid[pos])
            pos = pos + 1
    stdscr.refresh()
```

Client: Main Loop

```
#
# PURPOSE:
# This is the main loop for the client.  It initially receives 'num_rows',
# 'num_cols', 'player_id', and 'grid' from the server.  That grid is then
# displayed on the curses screen. Then, repeatedly, the client will wait
# for a message from the server, either to grant the current Player a turn,
# or to display the updated game board on the screen. If the Player is
# granted a turn, this loop will wait for a character input and then send
# that to the server.
#
# PARAMETERS:
# stdscr: Must be a reference to a valid curses screen
#
# RETURN/SIDE EFFECTS:
# N/A
#
# NOTES:
# Multiple character inputs will be buffered and transmitted at the rate of
# one character per Player turn. If the server rejects a move for being
# invalid, the Player will lose a turn.
# Exceptions will be caught and logged, and result in the orderly termination
# of this function.
#
```

Client: Main Loop

```
def main(stdscr):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        sock.connect((HOST, PORT))
        sock.sendall(ics226.HELLO)

        reply = ics226.getBuf(sock, 1)
        num_rows = struct.unpack('!B', reply)[0]

        reply = ics226.getBuf(sock, 1)
        num_cols = struct.unpack('!B', reply)[0]

        reply = ics226.getBuf(sock, 1)
        player_id = struct.unpack('!B', reply)[0]

        reply = ics226.getBuf(sock, num_rows * num_cols)
        grid = reply.decode('utf-8')

        game_env = Game_Env(num_rows, num_cols, str(player_id), grid)

        displayGrid(stdscr, game_env)
        logger.debug(game_env.player_id + ' is ready')
```

Client: Main Loop

```
    while True:
        data = ics226.getBuf(sock, 1)
        if data == ics226.GO:
            logger.debug(game_env.player_id + ' going ahead')
            displayGrid(stdscr, game_env)
            k = stdscr.getkey()
            sock.sendall(k.encode('utf-8'))
        elif data == ics226.GRID:
            logger.debug(game_env.player_id + ' got grid')
            reply = ics226.getBuf(sock, num_rows * num_cols)
            game_env.grid = reply.decode('utf-8')
            displayGrid(stdscr, game_env)
        elif data == ics226.QUIT:
            logger.debug(game_env.player_id + ' is quitting')
            break
except Exception as e:
    logger.critical(str(e), exc_info = 1)
sock.close()
```

Client: Main Loop

```
logger = logging.getLogger('client.py')
logger.setLevel(logging.DEBUG)
handler = logging.handlers.SysLogHandler(address = '/dev/log')
logger.addHandler(handler)

curses.wrapper(main)
```

Server: Setup

```
BLANK = '_'
MAX_PLAYERS = 2
NUM_COLS = 20
NUM_ROWS = 10
NUM_TREASURE = 10

# Format: [(row, col, label), (row, col, label), ...]
# Must contain exactly MAX_PLAYERS tuples
PLAYERS = [(0, 0, 'X'), (NUM_ROWS - 1, NUM_COLS - 1, 'Y')]

PORT = 12345
SPACER = 2
TREASURE = '$'

locks = []
for i in range(MAX_PLAYERS):
    locks.append(threading.Semaphore())
    locks[-1].acquire()
```

Server: Game_Env

```
#
# PURPOSE:
# A simple container for storing the values 'curr_player', 'grid',
# 'player_positions', and 'packed_grid'
#
# PARAMETERS:
# grid: A 2D array of characters representing the game board
# player_positions: An array of tuples in the form of (row, col, label)
# representing player positions and IDs
#
# RETURN/SIDE EFFECTS:
# N/A
#
# NOTES:
# No validation takes places
#
class Game_Env:
    def __init__(self, grid, player_positions):
        self.curr_player = 0
        self.grid = grid
        self.player_positions = player_positions
        self.packed_grid = ''
```

Server: addPlayersToGrid

```
#
# PURPOSE:
# Given a valid 'game_env', adds all players in 'game_env.player_positions'
# to 'game_env.grid' at the positions indicated in
# 'game_env.player_positions'
#
# PARAMETERS:
# game_env: The game environment; see Game_Env for details
#
# RETURN/SIDE EFFECTS:
# 'game_env' is updated as described above
#
# NOTES:
# N/A
#
def addPlayersToGrid(game_env):
    for row, col, label in game_env.player_positions:
        game_env.grid[row][col] = label
```

Server: addTreasureToGrid

```
#
# PURPOSE:
# Given a valid 'game_env', randomly add exactly 'NUM_TREASURE' treasures to
# 'game_env.grid'. Only 'BLANK' spaces will be replaced by 'TREASURE'
# characters.
#
# PARAMETERS:
# game_env: The game environment; see Game_Env for details
#
# RETURN/SIDE EFFECTS:
# 'game_env' is updated as described above
#
# NOTES:
# N/A
#
def addTreasureToGrid(game_env):
    i = NUM_TREASURE
    while i > 0:
        row = random.randint(0, NUM_ROWS - 1)
        col = random.randint(0, NUM_COLS - 1)
        if game_env.grid[row][col] == BLANK:
            game_env.grid[row][col] = TREASURE
            i = i - 1
```

Server: drawGrid

```
#
# PURPOSE:
# Given a valid curses screen 'stdscr' and a valid 'game_env', clears the
screen,
# then displays the grid on the curses screen.  At the same time, this
function
# updates 'game_env.packed_grid' to represents this grid in the format of a
string
# that can then be transmitted over the network
#
# PARAMETERS:
# stdscr: A curses screen reference
# game_env: The game environment; see Game_Env for details
#
# RETURN/SIDE EFFECTS:
# 'game_env' is updated as described above
#
# NOTES:
# N/A
#
```

Server: drawGrid

```
def drawGrid(stdscr, game_env):
    stdscr.clear()
    game_env.packed_grid = ''
    for row in range(NUM_ROWS):
        for col in range(NUM_COLS):
            stdscr.addstr(row, col * SPACER, game_env.grid[row][col])
            game_env.packed_grid = game_env.packed_grid + game_env.grid[row][col]
    stdscr.refresh()
```

Server: contactPlayer

```
#
# PURPOSE:
# This function is called by multiple threads (one thread per Player).
# Given a valid curses screen 'stdscr', TCP network socket 'sock', the ID
# of the player 'player_id', and the game environment 'game_env', initially
# waits for a connection from the Player and then sends out the number of
# rows, number of columns, player id, and game board. Thereafter, repeatedly,
# will attempt to get access to a critical region shared with all other
# Player threads.  Once access is granted, will send the grid to the player
# and then request the player to move to another position.  Once received,
# the move is validated.  If the move is not obstructed by a wall, treasure,
# or another player, it is accepted and the grid is updated accordingly.
# If the move is not valid, the player loses the turn.  The updated grid is
# then transmitted, and the critical region is vacated for the next player.
#
# PARAMETERS:
# stdscr: A curses screen reference
# sock: A valid TCP socket from which data can be received and to which
# data can be sent
# player_id: The ID of the player associated with this thread instance of
# 'contactPlayer'
# game_env: The game environment; see Game_Env for details
#
```

Server: contactPlayer

```
# RETURN/SIDE EFFECTS:
# 'game_env' is updated as described above
#
# NOTES:
# Exceptions will be caught and logged, and result in the orderly termination
# of this function.
#
def contactPlayer(stdscr, sock, player_id, game_env):
    player_id_str = str(player_id)
    try:
        sc, sockname = sock.accept()
        logger.debug('Acquired player ' + player_id_str + ' aka ' + str(sockname))
        data = ics226.getBuf(sc, 1).decode('utf-8')
        logger.debug('Got ' + data + ' from ' + player_id_str)
        sc.sendall(struct.pack('!B', NUM_ROWS))
        sc.sendall(struct.pack('!B', NUM_COLS))
        sc.sendall(struct.pack('!B', player_id))
        sc.sendall(game_env.packed_grid.encode('utf-8'))
```

Server: contactPlayer

```
while True:
    locks[player_id].acquire()
    logger.debug('Notifying ' + player_id_str)
    sc.sendall(ics226.GRID)
    sc.sendall(game_env.packed_grid.encode('utf-8'))
    sc.sendall(ics226.GO)
    logger.debug('Waiting for ' + player_id_str)
    data = ics226.getBuf(sc, 1).decode('utf-8')
    logger.debug('Got ' + data + ' from ' + player_id_str)
    (row, col, label) = game_env.player_positions[game_env.curr_player]
    game_env.grid[row][col] = BLANK
    if data == ics226.LEFT:
        if col > 0 and game_env.grid[row][col - 1] == BLANK:
            col = col - 1
    elif data == ics226.RIGHT:
        if col < NUM_COLS - 1 and game_env.grid[row][col + 1] == BLANK:
            col = col + 1
    elif data == ics226.UP:
        if row > 0 and game_env.grid[row - 1][col] == BLANK:
            row = row - 1
    elif data == ics226.DOWN:
        if row < NUM_ROWS - 1 and game_env.grid[row + 1][col] == BLANK:
            row = row + 1
    game_env.player_positions[game_env.curr_player] = (row, col, label)
```

Server: contactPlayer

```
            logger.debug('Placed ' + player_id_str + ' at ' +
str(game_env.player_positions[game_env.curr_player]))
            addPlayersToGrid(game_env)
            drawGrid(stdscr, game_env)
            sc.sendall(ics226.GRID)
            sc.sendall(game_env.packed_grid.encode('utf-8'))
            game_env.curr_player = (game_env.curr_player + 1) % MAX_PLAYERS
            locks[game_env.curr_player].release()
    except Exception as e:
        logger.critical(str(e), exc_info = 1)
    sc.close()
```

Server: main

```
# PURPOSE:
# Sets up a wildcard server socket at port 'PORT'. Then creates 'MAX_PLAYERS'
# threads, each of which will be running an instance of 'contactPlayer'
#
# PARAMETERS:
# stdscr: A curses screen reference
#
# RETURN/SIDE EFFECTS:
# N/A
#
# NOTES:
# Pressing a key will terminate this program; clients will not be notified
# Excpetions will be caught and logged and result in the termination of the game
#
def main(stdscr):
    try:
        game_env = Game_Env([[BLANK] * NUM_COLS for _ in range(NUM_ROWS)],
PLAYERS)
        addPlayersToGrid(game_env)
        addTreasureToGrid(game_env)
        drawGrid(stdscr, game_env)
```

Server: main

```
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        sock.bind(('', PORT))
        sock.listen(MAX_PLAYERS)
        num_players = 0
        threads = []
        while num_players != MAX_PLAYERS:
            threads.append(threading.Thread(target = contactPlayer, args = 
(stdscr, sock, num_players, game_env, )))
            threads[-1].start()
            num_players = num_players + 1
        locks[0].release()
        stdscr.getkey()
        sock.close()
    except Exception as e:
        logger.critical(str(e), exc_info = 1)
    finally:
        os._exit(os.EX_OK)

logger = logging.getLogger('server.py')
logger.setLevel(logging.DEBUG)
handler = logging.handlers.SysLogHandler(address = '/dev/log')
logger.addHandler(handler)
```