**GROUP MEMBERS NAMES:**

**15BCB0055 : AISHWARYA.S**

**15BCB0061 : GOUTHAMI REDDY**

**SLOT: D2**

**PARALLEL AND DISTRIBUTED COMPUTING**

**REVIEW 2**

**REAL TIME FAST CHESS**

**ABSTRACT:**

Chess is an interesting a challenging game played between two people. While playing the game, the players are need to make moves. The development of an online application can make it possible for the players to make moves even if they are not physically present together. The two players should be able to register for the game play. When one player makes a move, an email should be sent to the other notifying him about the move and asking to play his. This way, the players can play the game whenever they have time and access to the internet. The classes must implement all chess rules and moves, like "castlings", "en passant" captures, stalemates in case the opponent has no longer a legal move and his king is not under attack

**FUNCTIONAL REQUIREMENTS:**

1. a. Pawn Pawns shall move one space forward, optionally two spaces forward on their opening move. Priority 1
2. b. Rook Rooks shall move vertically or horizontally any number of spaces unless impeded by another piece. Priority 1
3. c. Knight Knights shall move two spaces either vertically or horizontally followed by one space perpendicularly. Priority 1
4. d. Bishop Bishops shall move diagonally any number of spaces unless impeded by another piece. Priority 1
5. e. Queen Queens shall move vertically, horizontally, or diagonally any number of spaces unless impeded by another piece. Priority 1
6. f. King Kings shall move one space in any direction. Priority 1
7. g. Castling When requirements are met for castling (see definition), kings may move two spaces towards a rook, with the rook moving onto the space crossed over by the king. Priority1
8. h. General Capture If a piece other than a pawn, moving in its normal fashion, may move into a square occupied by an opposing piece, the friendly piece may capture the opposing piece. Priority 1
9. i. Pawn Pawns shall capture by moving forward one space diagonally into an opposing piece. Priority 1
10. j. En Passant When requirements are met for en passant capture, a pawn may capture as above into a space crossed, but no longer occupied by an opposing piece. Priority 1
11. k. Promotion A pawn, having entered the rank opposite where it started, shall be promoted to a piece of its controller’s choosing. Priority 1
12. l. Legality A move shall be deemed illegal if it does not follow the above rules or would cause the moving player’s king to become in check. Priority 1

**NON – FUNCTIONAL REQUIREMENTS**

1. a. Establishing Connection shall be between two computers, each with the Chess program. Priority 1
2. b. IP Addresses Connections shall be determined by IP(v4) addresses. Priority 1
3. c. Game Start When two games have connected, they shall send a ready message to signal the start of the game. Priority 1
4. d. Protocol Messages shall be passed using algebraic chess notation. Priority 1

NON – FUNCTIONAL REQUIREMENTS

1. e. Establishing Connection shall be between two computers, each with the Chess program. Priority 1
2. f. IP Addresses Connections shall be determined by IP(v4) addresses. Priority 1
3. g. Game Start When two games have connected, they shall send a ready message to signal the start of the game. Priority 1
4. h. Protocol Messages shall be passed using algebraic chess notation. Priority 1

**OPERATIONAL REQUIREMENTS**

1. a. The network shall be available. Priority 1
2. b. Malformed File A file that does not follow the .sav format will be rejected and an error message displayed. Priority 2
3. c. Mid-game Failure If the network connection fails in the middle of a game, WRC shall retry 3 times and then display an error message. Priority 2
4. d. Bandwidth There shall be sufficient bandwidth to handle game messages. Priority 1
5. **Modules discussion:**
6. **Game design:**
7. These includes the toxonamy and analysing the previous work done by others.ensuring innovative ideas and bringing them out.this includes creativity technique and story boards.
8. **Game technology:**
9. These include how to create this system of online gaming.in which details of
10. HTML5; client scripting; 2D, 3D Graphics for Games; Sprites; Game Objects; Game Engine APIs; Animation; Collision Detection; Fundamental Physics; Fundamental AI; Optimisation and Performance should be known
11. Multiplayer:
12. This includes how the game are connected to each Network architecture; Client Server; Distributed; Peer to Peer; Protocols and coding part of the game.
13. **Game block:**
14. Should know about the details of and implement the game .finally should pass the edge of
15. Test Plan, Game Playability, Scalable Deployment, Marketing.

**Future enhancement:**

* • Use REDIS Cache for variables so the server can auto-scale beyond one instance
* • NOSQL backend to store games and players
* • Facebook authentication
* • Wrap the UI in a better framework like Ionic
* • Ship on mobile platforms using Cordova
* • Mobile push notifications
* • Consider building a 3D interface in Unity
* • Add some hardware component, because hardware

**CODE:**

var express = require('express');

var app = express();

app.use(express.static('public'));

app.use(express.static('dashboard'));

var http = require('http').Server(app);

var io = require('socket.io')(http);

var port = process.env.PORT || 3000;

const passport=require('passport')

const passportlocal=require('passport-local')

const session=require('cookie-session')

const bodyParser=require('body-parser')

const cookieParser=require('cookie-parser')

const student=require('./model/model.js')

const mongoose=require('mongoose')

const path=require('path')

const flash = require('connect-flash');

app.engine('ejs', require('ejs').renderFile);

app.set('view engine', 'ejs');

app.set('views', \_\_dirname + '/public');

mongoose.connect("mongodb://localhost/devfest-portal")

app.use(session({name:'session',keys:['aish','satish']}));

app.use(passport.initialize());

app.use(passport.session());

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }))

app.use(cookieParser());

var initPassport=require('./Authentication/init');

initPassport(passport);

var isAuthenticated = function (req, res, next) {

// if user is authenticated in the session, call the next() to call the next request handler

// Passport adds this method to request object. A middleware is allowed to add properties to

// request and response objects

if (req.isAuthenticated())

return next();

// if the user is not authenticated then redirect him to the login page

res.redirect('/login');

}

var lobbyUsers = {};

var users = {};

var activeGames = {};

app.get('/', function(req, res) {

res.render('home',{user: req.user});

});

app.get('/home', function(req, res) {

res.render('home' , {user: req.user});

});

app.get('/game', isAuthenticated, function(req, res) {

res.sendFile(\_\_dirname + '/public/index.html');

});

app.get('/login', function(req, res) {

res.sendFile(\_\_dirname + '/public/login.html');

});

app.post('/login', passport.authenticate('login', {

successRedirect: '/home',

failureRedirect: '/error',

failureFlash : true

}));

app.get('/signup', function(req, res) {

res.sendFile(\_\_dirname + '/public/signup.html');

});

app.post('/signup', passport.authenticate('signup', {

successRedirect: '/home',

failureRedirect: '/error',

failureFlash : true

}));

app.get('/dashboard/', function(req, res) {

res.sendFile(\_\_dirname + '/dashboard/dashboard.html');

});

io.on('connection', function(socket) {

console.log('new connection ' + socket);

socket.on('login', function(userId) {

console.log(userId + ' joining lobby');

socket.userId = userId;

if (!users[userId]) {

console.log('creating new user');

users[userId] = {userId: socket.userId, games:{}};

} else {

console.log('user found!');

Object.keys(users[userId].games).forEach(function(gameId) {

console.log('gameid - ' + gameId);

});

}

socket.emit('login', {users: Object.keys(lobbyUsers),

games: Object.keys(users[userId].games)});

lobbyUsers[userId] = socket;

socket.broadcast.emit('joinlobby', socket.userId);

});

socket.on('invite', function(opponentId) {

console.log('got an invite from: ' + socket.userId + ' --> ' + opponentId);

socket.broadcast.emit('leavelobby', socket.userId);

socket.broadcast.emit('leavelobby', opponentId);

var game = {

id: Math.floor((Math.random() \* 100) + 1),

board: null,

users: {white: socket.userId, black: opponentId}

};

socket.gameId = game.id;

activeGames[game.id] = game;

users[game.users.white].games[game.id] = game.id;

users[game.users.black].games[game.id] = game.id;

console.log('starting game: ' + game.id);

lobbyUsers[game.users.white].emit('joingame', {game: game, color: 'white'});

lobbyUsers[game.users.black].emit('joingame', {game: game, color: 'black'});

delete lobbyUsers[game.users.white];

delete lobbyUsers[game.users.black];

socket.broadcast.emit('gameadd', {gameId: game.id, gameState:game});

});

socket.on('chat message', function(msg){

io.emit('chat message', msg);

});

socket.on('resumegame', function(gameId) {

console.log('ready to resume game: ' + gameId);

socket.gameId = gameId;

var game = activeGames[gameId];

users[game.users.white].games[game.id] = game.id;

users[game.users.black].games[game.id] = game.id;

console.log('resuming game: ' + game.id);

if (lobbyUsers[game.users.white]) {

lobbyUsers[game.users.white].emit('joingame', {game: game, color: 'white'});

delete lobbyUsers[game.users.white];

}

if (lobbyUsers[game.users.black]) {

lobbyUsers[game.users.black] &&

lobbyUsers[game.users.black].emit('joingame', {game: game, color: 'black'});

delete lobbyUsers[game.users.black];

}

});

socket.on('move', function(msg) {

socket.broadcast.emit('move', msg);

activeGames[msg.gameId].board = msg.board;

console.log(msg);

});

socket.on('disconnect', function(msg) {

console.log(msg);

if (socket && socket.userId && socket.gameId) {

console.log(socket.userId + ' disconnected');

console.log(socket.gameId + ' disconnected');

}

delete lobbyUsers[socket.userId];

socket.broadcast.emit('logout', {

userId: socket.userId,

gameId: socket.gameId

});

});

/////////////////////

// Dashboard messages

/////////////////////

socket.on('dashboardlogin', function() {

console.log('dashboard joined');

socket.emit('dashboardlogin', {games: activeGames});

});

});

app.get('/signout', function(req, res) {

req.logout();

res.redirect('/home');

});

http.listen(port, function() {

console.log('listening on \*: ' + port);

});