2048 Game

This is a report for a game solver program. The objective is to get at least a 2048 tile (each position with a value is called tile), qualified as victory.

The algorithm used is a variation of minimax called expectiminimax[1]. In the minimizing player turn it calculates the average of all possibilities instead of picking the worst case, and the maximizing player just choose the maximum value as usual.

The language used to make this project is JavaScript, so the program is to be run in the website of the game ([3]), like it is a real player.

As the games takes lots of moves (about 600) and the algorithm grows exponentially it is not possible to search the all game tree in a fair amount of time, so an evaluation function has to be used. Based in the comment by “ovolve” in [1], the main condition user to evaluate each board is the monotonicity, meaning the tiles should grow only to the direction of one corner.

Two other criteria were used. The first is that there should not be many tiles with the same number especially if they have a low value (high values are the goal), and is computed by the following function:

The second is that it is a good idea to merge tiles, even more if they have a high value, computed by:

As the function grows squarely to the value a single four worth more than a couple of two’s. These functions were made using a lot of test, and put together using empirical numeric multiplying constants.

The result is that the program solved the game 9 out of 10 tests, achieving even 4096 two times, so it was very successful.

References

[1]-<http://en.wikipedia.org/wiki/Expectiminimax_tree>, Expectiminimax tree. Access 12/18/14.

[2]-<http://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048>, Stack Overflow 2048 thread. Access 12/18/14.

[3]- <http://gabrielecirulli.github.io/2048/>, Game website. Access 12/18/14.

Appendix A – Code

The following code can be used just by copping and pasting it at the website console.

|  |
| --- |
| // website to run: http://gabrielecirulli.github.io/2048/  GameSolver = function() {  this.loadjQuery();  this.run = false;  this.stop = false;  }  GameSolver.prototype.possibleGrids = function(grid){  var possibles = [];  for (var i=0; i<4; i++){  for (var j=0; j<4; j++){  if (grid[i][j] == null){  var game\_1 = this.copyGrid(grid);  var game\_2 = this.copyGrid(grid);  game\_1[i][j] = 2;  game\_2[i][j] = 4;  possibles.push([game\_1, 0.9]);  possibles.push([game\_2, 0.1]);  }  }  }  return possibles;  }  GameSolver.prototype.possibleMoves = function(grid\_p){  var moves = [];  for (var i=0; i<4; i++){  var grid = this.moveToSide(grid\_p, i);  if(!this.gridsAreEqual(grid\_p, grid))  moves.push([grid, i]);  }  return moves;  }  GameSolver.prototype.moveToSide = function(grid, side){  // 0: up, 1: right, 2: down, 3: left  var new\_grid = this.copyGrid(grid);  var merged = [[false, false, false, false], [false, false, false, false], [false, false, false, false], [false, false, false, false]];  switch (side){  case 0:  for (var i=0; i<4; i++) for (var j=1; j<4; j++) if(new\_grid[i][j] != null) for (var k=j; k>0; k--) {  if (new\_grid[i][k-1] == null){  new\_grid[i][k-1] = new\_grid[i][k];  new\_grid[i][k] = null;  merged[i][k-1] = merged[i][k];  merged[i][k] = false;  }  else if (new\_grid[i][k-1] == new\_grid[i][k] && merged[i][k] == false && merged[i][k-1] == false){  merged[i][k-1] = true;  new\_grid[i][k-1] \*= 2;  new\_grid[i][k] = null;  }  else break;  }  break;  case 1:  for (var j=0; j<4; j++) for (var i=2; i>=0; i--) if(new\_grid[i][j] != null) for (var k=i; k<3; k++) {  if (new\_grid[k+1][j] == null){  new\_grid[k+1][j] = new\_grid[k][j];  new\_grid[k][j] = null;  merged[k+1][j] = merged[k][j];  merged[k][j] = false;  }  else if (new\_grid[k+1][j] == new\_grid[k][j] && merged[k][j] == false && merged[k+1][j] == false){  merged[k+1][j] = true;  new\_grid[k+1][j] \*= 2;  new\_grid[k][j] = null;  }  else break;  }  break;  case 2:  for (var i=0; i<4; i++) for (var j=2; j>=0; j--) if(new\_grid[i][j] != null) for (var k=j; k<3; k++) {  if (new\_grid[i][k+1] == null){  new\_grid[i][k+1] = new\_grid[i][k];  new\_grid[i][k] = null;  merged[i][k+1] = merged[i][k];  merged[i][k] = false;  }  else if (new\_grid[i][k+1] == new\_grid[i][k] && merged[i][k] == false && merged[i][k+1] == false){  merged[i][k+1] = true;  new\_grid[i][k+1] \*= 2;  new\_grid[i][k] = null;  }  else break;  }  break;  case 3:  for (var j=0; j<4; j++) for (var i=1; i<4; i++) if(new\_grid[i][j] != null) for (var k=i; k>0; k--) {  if (new\_grid[k-1][j] == null){  new\_grid[k-1][j] = new\_grid[k][j];  new\_grid[k][j] = null;  merged[k-1][j] = merged[k][j];  merged[k][j] = false;  }  else if (new\_grid[k-1][j] == new\_grid[k][j] && merged[k][j] == false && merged[k-1][j] == false){  merged[k-1][j] = true;  new\_grid[k-1][j] \*= 2;  new\_grid[k][j] = null;  }  else break;  }  break;  }  return new\_grid;  }  GameSolver.prototype.minimax = function(grid, depth, maxPlayer){  if (depth == 0)  return this.heuristic(grid);  if (this.gridIsTerminal(grid))  return -100000000000000;  if (maxPlayer){  var children = this.possibleMoves(grid);  var alpha = Number.NEGATIVE\_INFINITY;  for (var i in children){  alpha = Math.max(alpha, this.minimax(children[i][0], depth-1, false));  }  return alpha;  }  else {  var children = this.possibleGrids(grid);  var acc = 0;  for (var i in children){  acc += children[i][1] \* this.minimax(children[i][0], depth-1, true);  }  return 2\*acc/children.length;  }  }  GameSolver.prototype.nextMove = function(\_grid){  var grid = this.convertGridFromOriginal(\_grid);  if (this.gridIsTerminal(grid)) return null;  var alpha = Number.NEGATIVE\_INFINITY;  var move = null;  var moves = this.possibleMoves(grid);  var empty\_tiles = 1;  for (var i=0; i<4; i++) for (var j=0; j<4; j++) if (grid[i][j] == null) empty\_tiles++;  for (var i=0; i<moves.length; i++){  var value = this.minimax(moves[i][0], 4, false);  if (value > alpha){  alpha = value;  move =moves[i][1];  }  }  return move;  }  GameSolver.prototype.heuristic = function(grid){  var points = 0;  var vals = this.existingValues(grid);  for (var i in vals) points -= 300\*(2048/i) \* vals[i]\* vals[i];  var merge\_points =0;  for (var i=0; i<4; i++) for (var j=0; j<4; j++) {  if (grid[i][j] != null) {  if(grid[i][j]<64) merge\_points+= grid[i][j]\*grid[i][j]/8;  else if(grid[i][j]<128) merge\_points += grid[i][j]\*grid[i][j];  else merge\_points+= 2\*grid[i][j]\*grid[i][j];  }  }  points += 100\*merge\_points;  points += 50\*this.monotonicityPoints(grid);  return points;  }  GameSolver.prototype.monotonicityPoints = function(grid){  var points=0;  for (var i=0; i<4; i++) {  var last\_found =null;  for (var j=0; j<4;j++){  if (grid[i][j]){  if (last\_found)  if (last\_found > grid[i][j]) points += 10\*last\_found\*Math.abs(last\_found - grid[i][j]);  last\_found = grid[i][j];  }  else {  points += 5\*last\_found\*last\_found;  }  }  }  for (var j=0; j<4; j++) {  var last\_found =null;  if(j==3) for (var i=0; i<4;i++){  if (grid[i][j]){  if (last\_found)  if (last\_found > grid[i][j]) points += 2\*last\_found\*Math.abs(last\_found - grid[i][j]);  last\_found = grid[i][j];  }  else {  points += last\_found\*last\_found;  }  }  else for (var i=3; i>=0;i--){  if (grid[i][j]){  if (last\_found)  if (last\_found > grid[i][j]) points += 2\*last\_found\*Math.abs(last\_found - grid[i][j]);  last\_found = grid[i][j];  }  else {  points += last\_found\*last\_found;  }  }  }  return -points;  }  GameSolver.prototype.existingValues = function(grid){  var vals = {};  for (var i = 0; i < 4; i++) {  for (var j = 0; j < 4; j++) {  if (grid[i][j]) {  if (vals[grid[i][j]]) vals[grid[i][j]]++;  else vals[grid[i][j]] = 1;  }  }  }  return vals;  }  GameSolver.prototype.gridIsTerminal = function (grid) {  for (var i = 0; i < 4; i++) {  for (var j = 0; j < 4; j++) {  if(grid[i][j] == null) return false;  }  }  for (var i = 0; i < 4; i++) {  for (var j = 0; j < 4; j++) {  if(i>0 && grid[i][j] == grid[i-1][j]) return false;  if(i<3 && grid[i][j] == grid[i+1][j]) return false;  if(j>0 && grid[i][j] == grid[i][j-1]) return false;  if(j<3 && grid[i][j] == grid[i][j+1]) return false;  }  }    return true;  };  GameSolver.prototype.gridsAreEqual = function(grid\_1, grid\_2){  for (var i=0; i<4; i++)  for (var j=0; j<4; j++)  if ((grid\_1[i][j]==null && grid\_2[i][j]!=null) || (grid\_2[i][j]==null && grid\_1[i][j]!=null) || (grid\_1[i][j]!=null && grid\_2[i][j]!=null && grid\_1[i][j]!=grid\_2[i][j]))  return false  return true;  }  GameSolver.prototype.copyGrid = function(grid){  var new\_grid = [];  for (var i=0; i<4; i++) new\_grid.push(grid[i].slice());  return new\_grid;  }  GameSolver.prototype.convertGridFromOriginal = function(grid){  var new\_grid = [];  for (var i=0; i<4; i++){  new\_grid.push([]);  for (var j=0; j<4; j++){  if (grid[i][j] == null)  new\_grid[i].push(null);  else  new\_grid[i].push(grid[i][j].value);  }  }  return new\_grid;  }  GameSolver.prototype.loadjQuery = function(){  var s=document.createElement('script');  s.setAttribute('src','https://code.jquery.com/jquery-2.1.1.min.js');  s.setAttribute('type','text/javascript');  document.getElementsByTagName('head')[0].appendChild(s);  }  window.requestAnimationFrame(function () {  window.gameManager = new GameManager(4, KeyboardInputManager, HTMLActuator, LocalStorageManager);  window.gameSolver = new GameSolver(); gameManager.keepPlaying = true;  var a= function(){  if (gameSolver.run) gameManager.move(gameSolver.nextMove(gameManager.grid.cells));  if (gameManager)  setTimeout(a, 20);  };  setTimeout(a, 1);  gameSolver.run = true;  }); |

Appendix B – Proposal

The project objective is to develop an agent to solve the 2048 game. The problem is to be solved using search, more precisely a variation of minimax algorithm to handle the stochastic factor of the game.

The implementation will be done in JavaScript, using the already built interface of the game for web browsers (<http://gabrielecirulli.github.io/2048/>).

A few steps will be followed to implement the agent:

1. Implement the search algorithm
2. Create a simple heuristic to test the algorithm
3. Improve the heuristic and depth of the search

The code will run in the web browser and will be evaluated simply by checking the score of the game, the higher the better (verifying the average value of a fair amount of tries).