**Class Path Finding.**

Pada tulisan sebelumnya kita telah membahas algorithma ini dalam bentuk gambar, sekarang kita membahas penerapannya dalam code.

Untuk menerapkan algorithma ini, kita akan membuat sebuah class bernama PathFinder.

Class ini memiliki beberapa variable dan method yang dijelaskan pada paragraph-paragraph berikutnya. Penjelasan class ini cukup panjang, karena banyak hal-hal yang harus dijabarkan. Saya akan menjelaskan class ini berdasarkan method-method yang tersedia dan berusaha untuk mengurutkannya sesuai dengan alur yang dibahas di tulisan sebelumnya. Dimulai dari method yang paling pertama dipanggil saat melakukan proses pencarian jalan hingga ke method terakhir yang menghasilkan jalur dari pencarian.

Dalam penjabaran class ini saya menerapkan metode Umum ke khusus. Dimana alur algorithma secara garis besar disusun dalam sebuah method di awal, kemudian dari method awal ini akan memanggil method-method lain yang akan menjabarkan alur logika ke yang lebih spesifik.

**Method find()**

Full syntax: find(sx: number, sy: number, tx: number, ty: number): Array<any>

Method ini adalah method awal yang dipanggil saat hendak mencari jalan. Method ini berfungsi sebagai inisialisasi dan isinya menggambarkan alur program secara garis besar.

Alurnya adalah sebagai berikut:

Pertama, bersihkan array \_cells. Array cells menampung semua cells yang dihasilkan dalam menjalankan algoritma ini. Kita harus membersihkannya setiap kali kita memulai pencarian baru, karena array ini kemungkinan akan berisi cell dari pencarian sebelumnya.

while (this.\_cells.length > 0) {

this.\_cells.pop();

}

Setiap item dari array ini berupa object yang bertipe PFCell.

Algorithma path-finding akan menghasilkan banyak sekali object bertipe PFCell, namun tidak semuanya akan dipakai. Method buildPath()-lah yang akan berfungsi untuk memilah dan menyusun cell-cell yang nantinya akan benar-benar dipakai sebagai jalur. Method ini akan dijelaskan di paragrah selanjutnya.

Setelah memberishkan array \_cells, kita memanggil method getPath(). Ini adalah method yang mencari jalan sebenarnya.

Method ini menghasilkan array bertipe PFCell dan disimpan dalam variable res.

res = this.getPath(sx, sy, tx, ty);

Hasil yang disimpan dalam variable res ini kemudian kita konvert ke array yang bertipe integer. Hal ini agar datanya lebih sederhana, lebih universal, dan mengurangi dependency terhadap class PFCell.

Hasil dari konvert ini kita simpan dalam variable resAr.

resAr = this.resToArray(res);

Setelah itu kita bersihkan variable res karena sudah tidak digunakan. Dan kita mereturn resAr sebagai hasil dari method ini.

while (res.length > 0) {

res.pop();

}

return resAr;

Dari method find() kita mendapatkan gambaran secara garis besar, dimana detailnya akan dijabarkan dalam method-method yang akan dijelaskan pada paragraph-paragraph berikut.

**Method getPath();**

Full syntax: getPath(sx: number, sy: number, tx: number, ty: number): Array<PFCell>

Method ini adalah method yang merinci alur pencarian dari method find(). Method ini menghasilkan array bertipe PFCell.

Alurnya adalah sebagai berikut:

Pertama kita check apakah posisi awal dan akhir adalah sama, bila ya maka pencarian kita hentikan, dan kita mereturn array kosong.

if ((sx == tx) && (sy == ty)) {

return res;

}

Bila pengecekan pertama lolos. Itu artinya posisi awal dan akhir tidak sama, maka kita lanjutkan dengan membuat cell pertama. Posisinya ada di posisi awal. Cell yang dibuat ini disimpan pada sebuah array \_cells. Seperti yang telah dijelaskan sebelumnya, array \_cells berguna untuk menampung semua cell-cell yang dihasilkan selama algorithma berlangsung.

Untuk membuat cell kita memanggil method cellCreate().

this.\_cells.push(this.cellCreate(null, sx, sy, tx, ty));

Setelah itu kita memulai sebuah perulangan untuk melakukan pencarian. Perulangan ini akan berjalan terus sampai jalur ditemukan, atau bila kondisi sudah tidak memungkinkan lagi untuk melanjutkan. Kondisi pencarian bisa dihentikan apabila jumlah cell yang dihasilkan sudah terlalu banyak, atau pencarian menemukan jalan buntu.

Pada awal perulangan ini kita melakukan pengecekan apakah jumlah cell yang dihasilkan sudah melebihi batas yang telah ditentukan. Kita menyimpan informasi batas maksimal cell dalam variable \_maxCells.

if ((this.\_cells.length >= this.\_maxCells))

Kita harus mengawasi jumlah cell agar tidak terlalu banyak dan menghabiskan memori. Bila jumlahnya terlalu banyak, dan kita belum sampai maka kita harus berhenti mencari jalan dan mereturn array kosong.

Bila ternyata kondisi jumlah cell masih kurang dari batas maksimal, maka kita melanjutkan untuk mencari jalan dengan cara membuat cell baru. Untuk itu kita memanggil method getOpenCell() yang akan mengembalikan cell yang masih terbuka.

cellCr = this.getOpenCell();

Bila cellCr bukan null, maka berarti masih ada cell yang masih terbuka dan bisa dikembangkan.

if (cellCr)

Kemudian kita menutup cell ini.

cellCr.open = false;

Kita memanggil method checkSampaiTujuan() untuk menguji apakah cell ini posisinya berada di posisi yang dituju. Bila hasilnya ya, berarti kita sudah sampai dan tidak perlu melanjutkan lagi.   
  
if (this.checkSampaiTujuan(cellCr.x, cellCr.y, tx, ty)) {

res.unshift(cellCr);

this.buildPath(cellCr, res);

return res;

}

Bila kita belum sampai ke tempat tujuan, maka kita memanggil cellOpen() yang akan membuat cell baru dari cell ini.

Bila ternyata tidak ada lagi cell yang masih terbuka, dan kita belum sampai, maka kita akhiri pencarian kita, dan mereturn array kosong [].

**Method cellCreate();**full syntax: cellCreate(parent: PFCell, i: number, j: number, targetX: number, targetY: number): PFCell

Method ini untuk membuat cell baru.

Saat membuat cell baru kita menentukan property-property sebagai berkut:

X: posisi horizontal

Y: posisi vertical

open: apakah cell ini berada dalam posisi open atau close. Status open artinya cell ini masih bisa dikembangkan.

Idx: index dari cell, kita akan memakai idx pada fungsi buildPath().

parent: apakah cell ini memiliki parent ataukah tidak, kalau ya, maka kita isi referensinya disini

dist: jarak dari cell ini ke target, jaraknya adalah jarak heuristic (perkiraan). Kita menghitungnya dengan cara menambahkan jarak vertikal + horizontal. Saat menghitung jarak ini, kita tidak perlu terlalu detail karena jaraknya hanyalah untuk perkiraan saja. Jarak ini nantinya kita gunakan sebagai perbandingan dengan cell yang lain, untuk menentukan cell mana yang terdekat.

**Method CellCreate()**

Full syntax: cellCreate(parent: PFCell, i: number, j: number, targetX: number, targetY: number): PFCell

Method ini dipanggil pada proses membuka cell baru. Pembuatan cell baru tidak melibatkan alur yang rumit. Kita membuat instance dari PFCell kemudian memasukkan semua properti yang dibutuhkan.

**Method BuildPath()**

Syntax: buildPath(cell: PFCell, res: Array<PFCell>): void

Method ini tidak mereturn apa-apa (void). Hasilnya disimpan dalam variable res yang juga berfungsi sebagai parameter.

Method ini berfungsi untuk untuk menyaring cell-cell yang dihasilkan selama algorithma berlangsung untuk menghasilkan cell yang memang benar-benar dipakai sebagai jalur karena dalam algorithma ini kita menghasilkan banyak cell yang tidak semuanya digunakan.

Penelusuran dimulai dari cell terakhir yang dihasilkan. Dari cell ini kita menelusuri parentnya hingga sampai ke cell yang pertama.

len = this.\_cells.length;

for (i = 0; i < len; i++) {

cellTemp = this.\_cells[i];

if (cell.parent && (cellTemp.idx == cell.parent.idx)) {

cellParent = cellTemp;

}

}

Bila parent tidak ketemua, maka penelusuran ini kita akhiri. Cell yang tidak memiliki parent atau idxnya -1 berarti dia adalah cell awal.

//parent gak ada, cell adalah cell awal, return;

if (cellParent == null) {

console.log("no parent");

return;

}

Bila parent ketemua maka kita simpan dalam array res.

res.unshift(cellParent);

Kemudian kita check apakah cell tersebut adalah cell awal. Bila ya maka penelurusan kita sudah selesai. Bila bukan cell awal, maka dari cell tersebut kita telusuri lagi parentnya hingga sampai ke cell awal.

if (cellParent.idx == -1) {

return

} else {

this.buildPath(cellParent, res);

}

Method ini bersifat rekursif, dimana dia akan memanggil dirinya sendiri sampai cell awal ketemu. Anda harus berhati-hati saat menggunakan method rekursif karena bisa berakibat method ini memanggil dirinya terus-menerus hingga tidak terbatas.

**Method getOpenCell()**

Method ini untuk mencari cell-cell yang masih terbuka dan jaraknya paling dekat ke target.

Pertama kita membuat mengasumsikan jarak terdekat adalah 10000. Angka ini bisa sebesar mungkin, agar bisa dibandingkan.

Kemudian kita periksa semua cell, apakah ada yang jaraknya kurang dari angka tersebut, dan cell tersebut adalah cell yang terbuka. Bila ya maka kita akan ambil cell tersebut sebagai cell hasil. Kita juga mengupdate jarak terdekat.

Kita teruskan memeriksa semua cell berikutnya. Hasil akhirnya adalah sebuah cell yang terbuka dan jaraknya paling dekat ke client

**Method checkSampaiTujuan()**

checkSampaiTujuan(i: number, j: number, tx: number, ty: number): Boolean

Method ini berfungsi untuk mengecek kita sudah sampai ke target apa belum. [referensi gambar]. Parameter I dan J adalah posisi sekarang, sedangkan tx dan ty adalah posisi target.

**Method cellOpen()**

dellOpen(cellCr: PFCell, tx: number, ty: number): void

Method ini berfungsi untuk membuka cell baru. Untuk jelasnya silahkan melihat kembali penjelasan sebelumnya. Kita membuka cell ke empat arah, yaitu atas, kanan, bawah dan kiri. Kita tidak membuka cell ke arah diagonal. Pembahasan mengenai pergerakan diagonal akan dibahas pada tulisan mendatang, inysa Allah.

Sebelum membuka cell baru, pertama kita mengecek dulu apakah kita bisa melakukannya atau tidak. Untuk itu kita memanggil method cellPossible(). Bila hasilnya true maka kita membuka cell baru dengan memanggil method cellCreate() dan menyimpan cell baru tersebut dalam array \_cells.

**Method cellPosPossible()**

cellPosPossible(ix: number, jx: number, cells: Array<PFCell>): Boolean

Method ini berfungsi untuk mengecek apakah kita bisa membuka cell baru di tempat tujuan. Syarat agar kita bisa membuat cell baru adalah pertama cell di tempat tersebut belum pernah dibuat, dan tempat tersebut tidak terblock. Untuk itu kita memanggil method cellExistsAtPos() dan method checkCanMoveToPos().

**Method checkCanMoveToPos()**

checkCanMoveToPos(x: number, y: number): boolean

Method ini berfungsi untuk mengecek apakah kita melangkah ke posisi tertentu di peta. Ada dua hal yang kita cek. Pertama apakah posisi kita di luar peta, dan kedua apakah tempat yang kita tuju di block atau tidak.

**Method cellExistsAtPos()**

cellExistsAtPos(ix: number, jx: number, cells: Array<PFCell>): boolean

Method ini berfungsi untuk mengecek apakah cell pada posisi tertentu ada atau tidak. Method ini dipanggil saat kita akan membuka cell baru. Kita mengecek apakah kita bisa membuka cell baru di posisi yang dilewatkan pada parameter ix dan jx.

Hal ini untuk menghindari kita membuat cell di tempat yang sama dua kali.

Kita memeriksa semua cell yang disimpan dalam variable cells yang bertipe Array<PFCell>. Bila cell memiliki cell posisi yang sama dengan ix dan jx, maka kita akan mereturn true.

Bila kita telah memeriksa semua cell dan tidak ada cell yang sama maka kita akan mereturn false.

**Method resToArray()**

resToArray(res: Array<PFCell>): Array<any>

Method ini berfungsi untuk menconvert hasil dari pencarian yang berupa Array bertipe PFCell ke array bertipe integer. Array bertipe integer lebih sederhana dari pada array bertipe PFCell.