Pactica 2 IA

Nivel **1**

La búsqueda de un camino valido hasta el objetivo se hace mediante un algoritmos de búsqueda en anchura. Me he decidido por este algoritmo puesto que el mapa mas grande es de 100*100 y no es un tamaño muy grande, por lo que el método de altura tarda 0,05 segundos mas o menos en obtener un camino valido.

Mi algoritmo busca primero un camino en casillas y luego este lo traduce a pasos a seguir por el agente.

```
bool ComportamientoJugador::pathFinding(const estado &origen, const estado &destino, list<Action> &plan) {
  bool encontradoDestino = buscarCaminoAnchura(origen, destino, plan, mapaResultado);
  // Descomentar para ver el plan en el mapa
  VisualizaPlan(origen, plan);
  return encontradoDestino;
}
```

La función buscarCaminoAnchura es la encargada de buscar el camino valido.

```
bool ComportamientoJugador::buscarCaminoAnchura(const estado &origen, const estado &destino, list<Action> &plan, const std::vector< std::vector< unsigned char> > &mapa){
  //Borro la lista
  plan.clear();
  int tamanoMapa = mapa.size();
  vector<vector<bool>> mapaCasillasVisitadas(tamanoMapa,vector<bool>(tamanoMapa, false));
  int incrementosFila[4] = {-1,0,1,0};
  int incrementosColumna[4] = {0,1,0,-1};
  //mirar si hay un aldeano enfrente
  if(obstaculoDelante){
    mapaCasillasVisitadas[ origen.fila + incrementosFila[origen.orientacion] ][ origen.columna + incrementosColumna[origen.orientacion] ] = true;
    obstaculoDelante = false;
  }
  //Creamos la cola para la busqueda en anchura
  queue <list<casillaMapa>> casillasAVisitar;
  //Preparamaos la lista necesaria para ir alamacenando el camino
  casillaMapa casillaInicio;
  casillaInicio = rellenarCasillaMapa(origen.fila, origen.columna);
  list<casillaMapa> caminoInicial,caminoActual;
  caminoInicial.push_back(casillaInicio);
  //iniciamos la cola de casillas a visitar con la lista en la que estado
  //La casilla de origen
  casillasAVisitar.push(caminoInicial);
  mapaCasillasVisitadas[origen.fila][origen.columna] = true;
  bool encontradoDestino = false;

  while (!casillasAVisitar.empty() && !encontradoDestino){
    caminoActual = casillasAVisitar.front();
    casillasAVisitar.pop();
    casillaMapa ultimaCasilla = caminoActual.back();

    if(ultimaCasilla.fila == destino.fila && ultimaCasilla.columna == destino.columna)
      encontradoDestino = true;
    else{

      casillaMapa casillaSiguiente;
      for(int i = 0; i < 4; i++){
        casillaSiguiente=rellenarCasillaMapa(ultimaCasilla.fila + incrementosFila[i],ultimaCasilla.columna + incrementosColumna[i]);
        if(esCasillaValida(casillaSiguiente,mapa))
          if(!mapaCasillasVisitadas[casillaSiguiente.fila][casillaSiguiente.columna]){
            mapaCasillasVisitadas[casillaSiguiente.fila][casillaSiguiente.columna] = true;
            caminoActual.push_back(casillaSiguiente);
            casillasAVisitar.push(caminoActual);
            caminoActual.pop_back();
          }
      }

    }
  }
  if(encontradoDestino)
    construirPlan(caminoActual,plan,origen.orientacion);

  return encontradoDestino;
}
```

La funciona construirPlan es la encargada de traducir el camino de casillas a pasos necesarios para llegar al punto.

```
void ComportamientoJugador::construirPlan( const list<casillaMapa> &camino, list<Action> &plan, int orientacionInicial){
  list<casillaMapa>::const_iterator it = camino.begin();
  casillaMapa actual, siguiente;

  actual = *it;
  ++it;
  int calculoFila, calculoColumna, orientacion;
  estado estadoAnterior = rellenarEstado(actual.fila, actual.columna, orientacionInicial);

  while(it != camino.end()){
    siguiente = *it;
    calculoFila = siguiente.fila - actual.fila;
    calculoColumna = siguiente.columna - actual.columna;
    if(calculoColumna == 0)
      if(calculoFila == -1)
        orientacion = 0;
      else
        orientacion = 2;
    else if( calculoColumna == -1)
        orientacion = 3;
    else
        orientacion = 1;

    //Si el estado anterior es dinstino al de ahora hay que girar
    if(estadoAnterior.orientacion != orientacion){

      int destino = (estadoAnterior.orientacion + 1) % 4;
      if( destino == orientacion){
        plan.push_back(actTURN_R);
        estadoAnterior.orientacion = destino;
      }else
        while( orientacion != estadoAnterior.orientacion ){
          estadoAnterior.orientacion = (estadoAnterior.orientacion + 3) % 4;
          plan.push_back(actTURN_L);
        }

    }
    estadoAnterior = rellenarEstado(siguiente.fila, siguiente.columna, estadoAnterior.orientacion);
    plan.push_back(actFORWARD);

    actual = siguiente;
    ++it;
  }

}
```

Nivel 2

En este nivel se añade al think el siguiente código que hace que cuando tengamos un aldeano en la casilla a la que el agente se aproxima, este lo detecte y recalcule una nueva ruta.

```
if (hayPlan && !esValidoAvanzar(sensores.terreno[2],sensores.superficie[2]) && plan.front() == 0){
  cout << "obstaculo enfrente\n";
  hayPlan = false;
  obstaculoDelante = true;
}
```

Poniendo hay plan a false lo que se hace es hacer que el agente tenga que recalcular nuevamente un camino y poniendo obstaculoDelante ,que es una nueva variable del agente añadida en este nivel, a true marca la casilla donde no podemos avanzar como ya visitada forzando que el método de anchura no la visite.

Nivel 3

El aldeano anda hacia delante hasta llegar a un obstáculo o hasta llegar a un máximo numero de paso (tamaño del mapa / 10) en ese momento decide si girar a izquierda o derecha con un aleatorio,

La detección de el pk se realiza mediante los sensores. Si se detecta el pk se transforman los sensores en una matriz y se realiza una búsqueda en anchura en esa matriz para buscar un camino hasta el pk. Una vez encontrado el pk se empieza a escribir el mapa mirando que la casilla no este escrita de antes.

A la hora de hacer el camino a el punto después de encontrar un pk se calcula teniendo en cuenta las ? Como un casilla valida y si llegado el momento no lo es al tenerla ya reescrita se recalcula el camino.

```
bool ComportamientoJugador::buscarCaminoPkSensores(Sensores sensores){
  bool encontradoDestino = true;
  int filasMatriz = 3 , columnasMatriz = 7;
  vector<vector<unsigned char>> matrizSensores(filasMatriz,vector<unsigned char>(columnasMatriz, 'X'));
  estado origen,destino;

  origen = rellenarEstado(2,3,0);

  int  posicionVector = 1, elementosColumna = 3 ;
  for (int i = 2; i >= 0; i-- ){
    int total = i +  elementosColumna;
    for(int j = i; j < total ; j++ ){
      matrizSensores[i][j] = sensores.terreno[posicionVector];
      if(sensores.terreno[posicionVector] == 'K'){
        destino.fila = i;
        destino.columna = j;
      }
      posicionVector++;
    }
    elementosColumna += 2;
  }
  for(int i = 0; i < filasMatriz; i++){
    for(int j = 0; j < columnasMatriz; j++)
      cout << matrizSensores[i][j] << " ";

    cout << endl;

  }
  encontradoDestino = buscarCaminoAnchura(origen, destino, plan, matrizSensores);
  if(encontradoDestino)
    plan.push_front(actFORWARD);


  return encontradoDestino;
}
```