**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по курсовому проекту**

**по дисциплине «Robot Operating System»**

Тема: **Футбол**

Студенты гр. 2304: Данилов С.В.

Комаров Д.Н.

Студент гр.2303: Козлов П.В.

Преподаватель: Филатов А. Ю.

Санкт-Петербург

2017

**Цель работы.**

Симулировать игру двух команд роботов с целью закатить мяч в чужие ворота

**Постановка задачи.**

Две команды, один мяч. Можно давать пас, можно вести мяч. Наезжать друг на друга нельзя. Можно отобрать мяч (как во время паса, так и у едущего с ним робота). Игра до победного гола (нескольких голов).

* В команде три-пять роботов, один из которых вратарь. Разрешено использование коллективного разума для принятия решения или наличие одного централизованного мозга.
* У роботов есть координаты других относительно друг друга, а также координаты мяча.

**Решение.**

*Алгоритм передвижения футболистов*

Для передвижения футболистов по карте используется алгоритм А\*(A star), так как он является лучшим для поиска кратчайшего пути до конечной точки.

А\* пошагово просматривает все пути, ведущие от начальной точки в конечную, пока не найдёт минимальный. Также как и жадный алгоритм является алгоритмом поиска по первому лучшему совпадению, однако A\* при выборе новой вершины учитывает весь пройденный до неё путь. Алгоритм сначала просматривает все узлы смежные с начальным, выбирается узел с минимальным путём. На каждом этапе алгоритм оперирует с множеством путей из начальной точки до всех ещё не раскрытых вершин графа.

*Структура проекта*

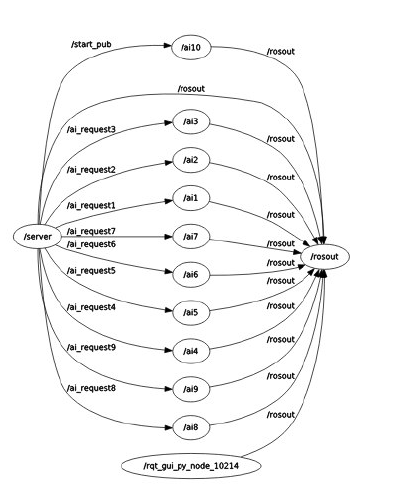


Рис. 1 - структура

На рис.1 показаны узлы проекта:

* aiN – узлы игроков, которые содержат логику перемещения игроков по полю(при помощи А\*), пинанию мяча и прочих свойственных футболистам действий. Подписываются на rosout для получения команд от пользователя.
* server – узел игры, подписывается на узлы aiN для отслеживания действий игроков и расчёта их положения на поле.
* rqt\_gui\_py\_nide\_10214 – узел для отрисовки графического интерфейса, подписывается на rosout для получения команд от пользователя.

**Реализация**

Полный код можно посмотреть тут - <https://github.com/SammyVimes/robo-soccer/tree/ros>

Ниже представлена реализация алгоритма А\*.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | Class AStar: | |  | def \_\_init\_\_(self, start\_pixel\_pos, target\_pixel\_pos, size, blocked\_rects=[]): | |  | ## Number of nodes horizontally and vertically: | |  | self.sizex, self.sizey = size[0], size[1] | |  | self.unit\_cost = 10 | |  | self.unit\_diagonal\_cost = 14 | |  | self.blocked\_poses = [] | |  |  | |  | ## Create start and target nodes: | |  | start\_node\_pos = self.to\_node\_pos(start\_pixel\_pos) | |  | target\_node\_pos = self.to\_node\_pos(target\_pixel\_pos) | |  | self.target\_node = Node(pos=target\_node\_pos) | |  | self.start\_node = Node(pos=start\_node\_pos, g=0, h=self.calc\_h(start\_node\_pos)) | |  |  | |  | #self.create\_all\_nodes() #[None] \* (self.sizex \* self.sizey) | |  | self.openlist = [] | |  | self.all\_nodes = {} | |  |  | |  | self.rects\_to\_poses(blocked\_rects) | |  | if not (self.start\_node.pos in self.blocked\_poses): | |  | self.openlist.append(self.start\_node) | |  | self.closedlist = [] | |  |  | |  | def rects\_to\_poses(self, rects): | |  | for rect in rects: | |  | topleft = self.to\_node\_pos(rect.topleft) | |  | bottomright = self.to\_node\_pos(rect.bottomright) | |  | width = int(abs(topleft[0] - bottomright[0])) | |  | height = int(abs(topleft[1] - bottomright[1])) | |  | for y in range(0, height+1): | |  | for x in range(0, width+1): | |  | self.blocked\_poses.append([topleft[0]+x,topleft[1]+y]) | |  |  | |  | def to\_node\_pos(self, pixel\_pos): | |  | return [math.floor(self.sizex \* (pixel\_pos[0] / general.width )), | |  | math.floor(self.sizey \* (pixel\_pos[1] / general.height))] | |  |  | |  | def to\_pixel\_pos(self, node\_pos): | |  | return [1.0 \* general.width \* node\_pos[0] / self.sizex + 1.0 \* (general.width/self.sizex)/2.0, | |  | 1.0 \* general.height \* node\_pos[1] / self.sizey + 1.0 \* (general.height/self.sizey)/2.0] | |  |  | |  | def draw\_blocks(self): | |  | for node in self.blocked\_poses: | |  | x = (1.0 \* node[0]/self.sizex) \* general.width + 1.0 \* general.width /self.sizex/2.0 | |  | y = (1.0 \* node[1]/self.sizey) \* general.height + 1.0 \* general.height/self.sizey/2.0 | |  | general.drawCircle(general.surface, pygame.Color(0,0,0), [x,y], 5,0) | |  |  | |  | def draw\_grid(self): | |  | for i in range(0, self.sizex): | |  | x = (1.0 \* i/self.sizex) \* general.width | |  | general.drawLine(general.surface, pygame.Color(0,0,0), [x,0], [x,general.height]) | |  | for i in range(0, self.sizey): | |  | y = (1.0 \* i/self.sizey) \* general.height | |  | general.drawLine(general.surface, pygame.Color(0,0,0), [0,y], [general.width,y]) | |  |  | |  | #def create\_all\_nodes(self): | |  | # all\_poses = [[x,y] for y in range(0,self.sizey) for x in range(0,self.sizex) ] | |  | # self.all\_nodes = [Node(pos=pos, g=0, h=self.calc\_h(pos)) for pos in all\_poses] | |  |  | |  | def get\_node(self, pos): | |  | pos\_tuple = tuple(pos) | |  | pos\_list = list(pos) | |  | if not pos\_tuple in self.all\_nodes: | |  | self.all\_nodes[pos\_tuple] = Node(pos=pos\_list, g=0, h=self.calc\_h(pos\_list)) | |  | return self.all\_nodes[pos\_tuple] | |  |  | |  | def get\_adjacents(self, node): | |  | ret = [] | |  | for y in range(int(node.pos[1]-1), int(node.pos[1]+1+1)): | |  | for x in range(int(node.pos[0]-1), int(node.pos[0]+1+1)): | |  | if not (x == node.pos[0] and y == node.pos[1]) and (x >= 0 and y >= 0) and (x < self.sizex and y < self.sizey) and not ([x,y] in self.blocked\_poses): | |  | adj\_node = self.get\_node([x,y]) | |  | if not (x == node.pos[0] or y == node.pos[1]): | |  | adj\_node.diagonal = True | |  | ret.append(adj\_node) | |  | return ret | |  |  | |  | def get\_astar\_paths(self): | |  | path = [] | |  |  | |  | if self.target\_node.pos in self.blocked\_poses: | |  | return path | |  | while len(self.openlist) > 0: | |  | min\_f = -1 | |  | next\_node = None | |  | for node in self.openlist: | |  | if node.f < min\_f or min\_f == -1: | |  | next\_node = node | |  | min\_f = node.f | |  |  | |  | if next\_node is None: | |  | print("node is None") | |  | return path | |  |  | |  | self.openlist.remove(next\_node) | |  | self.closedlist.append(next\_node) | |  |  | |  | for node in self.get\_adjacents(next\_node): | |  | if node in self.closedlist: | |  | continue | |  | if not (node in self.openlist): | |  | node.parent = next\_node | |  | node.g = node.parent.g | |  | if node.diagonal: node.g += self.unit\_diagonal\_cost | |  | else: node.g += self.unit\_cost | |  | node.f = node.g + node.h | |  | self.openlist.append(node) | |  | else: | |  | new\_cost = next\_node.g | |  | if node.diagonal: new\_cost += self.unit\_diagonal\_cost | |  | else: new\_cost += self.unit\_cost | |  | if node.g > new\_cost: | |  | node.parent = next\_node | |  | node.g = new\_cost | |  | node.f = node.g + node.h | |  |  | |  | path.append(next\_node) | |  | if next\_node.pos == self.target\_node.pos: | |  | return path | |  | return path | |  |  | |  | def get\_shortest\_path(self): | |  | paths = self.get\_astar\_paths() | |  | if paths == []: | |  | return [] | |  | node = paths.pop() | |  | shortest\_path = [] | |  | while node != None: | |  | shortest\_path.append(self.to\_pixel\_pos(node.pos)) | |  | node = node.parent | |  | shortest\_path.reverse() | |  | return shortest\_path | |  |  | |  | def draw\_path(self, path, node): | |  | if node == None: | |  | return path | |  | else: | |  | path.append(node) | |  | #general.drawCircle(general.surface, pygame.Color(255,0,0), self.to\_pixel\_pos(node.pos), 5,0) | |  | #self.draw\_path(node.parent) | |  |  | |  | def calc\_f(self, node): | |  | node.f = node.g + node.h | |  | return node.f | |  |  | |  | def calc\_h(self, node\_pos): | |  | h = self.unit\_cost \* (general.diff(node\_pos[0], self.target\_node.pos[0]) +\ | |  | general.diff(node\_pos[1], self.target\_node.pos[1]) ) | |  |  | |  | #xDistance = abs(node\_pos[0]-self.target\_node.pos[0]) | |  | #yDistance = abs(node\_pos[1]-self.target\_node.pos[1]) | |  | #if xDistance > yDistance: | |  | # h = 14\*yDistance + 10\*(xDistance-yDistance) | |  | #else: | |  | # h = 14\*xDistance + 10\*(yDistance-xDistance) | |  |  | |  | #print node\_pos, self.target\_node.pos, ret | |  | return h | |

**Выводы**

В результате курсовой работы была симулирована игра двух команд с целью закатить мяч в чужие ворота. Для построения маршрута игроков был использован алгоритм А\*.