01/01/2016

AI Project
-Vindinium-

Mustafa Safa Özdayı, 150110128
Taha Emre, 150120119

Table of Contents

1. Introduction

This report is prepared in order to explain the AI logic behind our bot, *Zerg2Win*, and as well as to provide some statistics related with it's competition result. Report is organized as follows: first, we will explain the methods we implemented and provide the related code. Second, we will provide data obtained from arena and training sessions. As last, we will reveal the logic behind our bot.

2. Implemented Methods

We started by implementing A* and it's helper methods such as manhattan distance (so, we used manhattan as our heuristic). After that, we implemented some helper methods to list places (mine/tavern/other heroes) by distance and also a method to get a valid direction (North, South etc.) in order to move to our path. We believe that the comments are sufficient to understand what the code does.

2.1 findPath() method

```
def findPath(self, start, goal):
    """Finds a path between start and goal using A*"""
    frontier = PriorityQueue()
    frontier.put(start, 0)
    came_from = {}
    cost_so_far = {}
    came_from[start] = None
    cost_so_far[start] = 0
    cost = 1

    while not frontier.empty():
      current = frontier.get()

      #2nd control is necessary for taverns&mines
      if current == goal or (goal in self.obstacles and self.manhattanDist(current, goal) == 1):
        data = []
        if current != goal:
          data.append(goal)
        while current in came_from:
          data.append(current)
          current = came_from[current]
        data.reverse()
        return data[1:]

      for node in self.getWalkableAdjacents(current):
        new_cost = cost_so_far[current] + cost
        if node not in cost_so_far or new_cost < cost_so_far[node]:
          cost_so_far[node] = new_cost
          priority = new_cost + self.manhattanDist(goal, node)
          frontier.put(node, priority)
          came_from[node] = current
    return None
```

2.2 getWalkableAdjacents() method

```
def getWalkableAdjacents(self, pos):
        """Returns walkable adjacent positions w.r.t to pos"""
        #1. Get adjacent nodes
        adjacents = []
        adjacents.append((pos[0], pos[1] - 1))
        adjacents.append((pos[0], pos[1] + 1))
        adjacents.append((pos[0] + 1, pos[1]))
        adjacents.append((pos[0] - 1, pos[1]))
        #2. Determine the walkable ones
        walkableAdjacents = []
        for node in adjacents:
            if node not in self.obstacles:
                if (node[0] >-1 and node[0] < self.game.board_size) and (node[1] >-1 and node[1] < 
self.game.board_size):
                    walkableAdjacents.append(node)
        return walkableAdjacents
```

2.3 manhattanDist() method

```
def manhattanDist(self, pos1, pos2):
        """Returns the manhattan distance between pos1 and pos2"""
        return abs(pos1[0] - pos2[0]) + abs(pos1[1] – pos2[1])
```

2.4 orderByDistance() method

```
def orderByDistance(self, pos, places):
        """ Orders places by distance w.r.t to pos """
        return sorted(places, key=lambda item:self.manhattanDist(pos, item)
```

2.5 getMove() method

```
def getMove(self, path_to_goal):
        """ Returns a valid command from one of [North, East, South, West, Stay] """
        if path_to_goal is None or len(path_to_goal) == 0:
            return "Stay"
        elif (len(path_to_goal) > 0):
            nextMove = path_to_goal[0]
            heroPos =  self.game.hero.pos
            dx = nextMove[1] - heroPos[1]
            dy = nextMove[0] - heroPos[0]
            if (dx == 0 and dy == 1):
                return "South"
            elif (dx == 0 and dy == -1):
                return "North"
            elif (dx == 1 and dy == 0):
                return "East"
            elif (dx == -1 and dy == 0):
                return "West"
            elif (dx == 0 and dy == 0):
                return "Stay"
```

3. Statistics

* When you click on links, you will see turn as 1200. This is the the total # of turns for 4 heroes. So, 300 turns for each hero.

3.1 Training Statistics

Map M1 (Symmetric 10x10), 300 Turns		
Run #	**Gold Gained**	**Game Link**
1	952	http://vindinium.org/brbe03qi
2	1050	http://vindinium.org/ulr2fd3n
3	971	http://vindinium.org/hcfl6b41
4	1029	http://vindinium.org/slbxymbs
5	1015	http://vindinium.org/7uojbtco
6	823	http://vindinium.org/7egzct67
7	876	http://vindinium.org/7heyvjsu
8	1079	http://vindinium.org/b12xerwp
9	993	http://vindinium.org/zrnykywe
10	1080	http://vindinium.org/tl0vwnx9

Average = 986.8
Standard Dev. = 84.56

Map M2 (Symmetric 12x12), 300 Turns		
Run #	**Gold Gained**	**Game Link**
1	1211	http://vindinium.org/en0j0mey
2	1291	http://vindinium.org/s87l445b
3	974	http://vindinium.org/39o13mbn
4	1173	http://vindinium.org/bimmulbt
5	1411	http://vindinium.org/waksnps5
6	1024	http://vindinium.org/wbivj26y
7	1601	http://vindinium.org/7b9j4fmi
8	1516	http://vindinium.org/waj75g3h
9	1209	http://vindinium.org/n4y06hwu
10	1194	http://vindinium.org/pb90msk2

Average = 1260.4
Standard Dev. = 199.99

Map M3 (Asymmetric 20x20), 300 Turns		
Run #	**Gold Gained**	**Game Link**
1	3078	http://vindinium.org/z5dghmck
2	2997	http://vindinium.org/h3k5oi51
3	2969	http://vindinium.org/ygcaak87
4	2985	http://vindinium.org/k5ksdr3e
5	2180	http://vindinium.org/4bqloz6g
6	3029	http://vindinium.org/tbu927il
7	2265	http://vindinium.org/6nolb0rk
8	2941	http://vindinium.org/54ync1vy
9	2960	http://vindinium.org/3ulne6zo
10	2980	http://vindinium.org/mntohxnu

Average = 2838.4
Standard Dev = 327.46

Map M4 (Symmetric 18x18), 300 Turns		
Run #	**Gold Gained**	**Game Link**
1	725	http://vindinium.org/xpd07egf
2	656	http://vindinium.org/04ehx9km
3	954	http://vindinium.org/rd59trpg
4	820	http://vindinium.org/1k5rq2bw
5	726	http://vindinium.org/3t7kldc0
6	994	http://vindinium.org/p0qjc7vz
7	802	http://vindinium.org/t6isp8cp
8	714	http://vindinium.org/rbj7caqj
9	1246	http://vindinium.org/ikajqgxi
10	749	http://vindinium.org/y7lzc8ce

Average = 838.6
Standard Dev = 178.7

Map M6 (Symmetric 12x12), 300 Turns		
Run #	**Gold Gained**	**Game Link**
1	776	http://vindinium.org/9m5n3b6g
2	564	http://vindinium.org/1ioqzsmv
3	518	http://vindinium.org/l4loev0o
4	620	http://vindinium.org/6km7q750
5	894	http://vindinium.org/e1z4ui0k
6	630	http://vindinium.org/0yj99f19
7	552	http://vindinium.org/pu7ttnn3
8	545	http://vindinium.org/d87t4toy
9	689	http://vindinium.org/ag65b0j9
10	629	http://vindinium.org/5fwchpi0

Average = 614.7
Standard Dev = 117.11

3.2 Arena Statistics

Arena, 300 Turns			
Run #	**Map Size**	**Gold Gained**	**Game Link**
1	12x12	47	http://vindinium.org/zhnh1mvg
2	10x10	284	http://vindinium.org/fymx0b9h
3	28x28	475	http://vindinium.org/8309p88s
4	22x22	601	http://vindinium.org/rcva3jjj
5	28x28	214	http://vindinium.org/y4sohygu
6	14x14	183	http://vindinium.org/lg2fdltc
7	22x22	601	http://vindinium.org/oa0yg7yn
8	12x12	110	http://vindinium.org/bkvk9ybn
9	28x28	1500	http://vindinium.org/y6dltx9z
10	24x24	503	http://vindinium.org/mjzea4ce

Average = 451.8
Standard Dev. = 419.5

4. AI Logic

The AI logic of our bot consist of 3 steps. Basically, it is as follows:

1. If hp is below 60 (can be killed with 3 hits) and have money to afford tavern, go to tavern
2. If the nearest enemy owns mine or mines and closer than the nearest available mine, go after him
3. Just go after the nearest available mine if above cases fail

The most valuable thing is the HP of our bot. So, we always tend to take care of that first. Then, we preferred to chase heroes that has mines if they are closer than the closest mine as we preferred to obtain mines by killing them rather than goblins. As last, if there is not a hero with at least a mine or if all of them are far away than the nearest available mine, (an available mine is a mine that is not owned by our hero) we sent our hero to conquer mines. The code piece that handles the logic can be seen below:

```
"""Game logic begins here"""
    #1. If hp is below 60 (can be killed with 3 hits) and have money to afford tavern, go to tavern
    if self.game.hero.life <= 60 and self.game.hero.gold >= 2:
        path_to_goal = self.findPath(self.game.hero.pos, nearest_tavern_pos)
        action = actions[1]
    #2. If the nearest enemy owns mine or mines and closer than the nearest available mine, go
after him
    else:
        for enemy in self.game.heroes:
            #the nearest enemy
            if enemy.pos == nearest_enemy_pos and (self.manhattanDist(self.game.hero.pos,
enemy.pos) <= self.manhattanDist(self.game.hero.pos, available_mines_by_distance[0])):
                if enemy.mine_count > 0:
                    path_to_goal = self.findPath(self.game.hero.pos, nearest_enemy_pos)
                    action = actions[2]
                    break
    #3. Just go after the nearest available mine if above cases fail
    if path_to_goal == []:
        path_to_goal = self.findPath(self.game.hero.pos,  available_mines_by_distance[0])
        action = actions[0]
"""Game logic ends here"""
```