Christopher LaJon Morgan

CS 470: BZRFlags Tutorial

Sep 17, 2013

**Really Dumb Agent**

Run

**Potential Fields Implementation and Description**

* Attractive Fields (Write Up and Implementation) Why and How
* Repulsive Fields (Write Up and Implementation) Why and How
* Tangential Fields (Write Up and Implementation) Why and How
* Graphs Of Potential Fields
  + Attractive
  + Repulsive
  + Tangential
  + Total
* Tuning Process

**You will need to test your PF agent as follows:**

1. Run your pf agent against your "Really Dumb Agent". Your pf agent should win, although you will have to do some tuning to win

Mods: Go after the closest flag. Switch to home goal after grabbing a flag.

1. Run against two dumb agents, you should win

The really dumb agents can get off some effective shots.

1. Run your pf agent against another copy of itself

Even though there is no AI to return ones flag, red was able to return their flag several times because then ended up on the same shortest distance path between their enemy’s base. Therefore the flag ended up in the line of travel for the red team.

1. Find another group, run your pf against their "Really Dumb Agent", you should win again

Overshot, more variance over a network..  
First of all I won! Second, one interesting observation is that the delay over the network cause my tank to slightly overshoot their goals. The tanks had a little too much velocity and would overshoot the flag slightly every time.

1. Run against two dumb agents from the other group, you should be able to win

Won like a boss.  
In all of my trials I never had friendly fire turn on. So, during the trials my tanks obliterate each other. I need a much more complicated firing algorithm.

1. Run your pf agent against their pf agent. Note that your grade does NOT depend upon which team wins, but it does depend upon what conclusions you can draw from the experience.

One complicated part of this project was translating tank coordinates into space that was continuous.   
Speed is and distance are big deals.

**Really Dumb Agent Complete Code**

|  |
| --- |
| import sys  import math  import time  from myPrint import \*  from bzrc import BZRC, Command  ###########################Potential Field Fun############################################  ####################################################################  # Distance between two points.  ####################################################################  def distance(x,y,goal):  return math.sqrt(((goal.y - y)\*(goal.y - y)) + ((goal.x - x)\*(goal.x - x)))  def distancePoints(x,y,xg,yg):  return math.sqrt(((yg - y)\*(yg - y)) + ((xg - x)\*(xg - x)))  def sign(a):  if a == 0 or a == -0:  return 0  return a / -a  ####################################################################  # Generate a Single Repulsive feild.  ####################################################################  def generateAnRepulsiveField(x,y, obsticle, makeItTangent=False, goal=None):  r = distancePoints(obsticle[0][0],  obsticle[0][1],  obsticle[2][0],  obsticle[2][1]) / 2.0  center = (obsticle[0][0] + ((obsticle[2][0] - obsticle[0][0]) / 2.0),  obsticle[0][1] + ((obsticle[2][1] - obsticle[0][1]) / 2.0))  s = 60.0  b = 1.0/s    d = distancePoints(x,y,center[0], center[1])  theta = math.atan2(center[1] - y, center[0] - x)    dx = -math.cos(theta)  dy = -math.sin(theta)    if makeItTangent:  thetaL = theta - (math.pi / 2.0)  thetaR = theta + (math.pi / 2.0)    dxL = -math.cos(thetaL)  dyL = -math.sin(thetaL)    dxR = -math.cos(thetaR)  dyR = -math.sin(thetaR)    if distancePoints(x + dxL, y + dyL, goal.x, goal.y) < distancePoints(x+dxR,y+dyR,goal.x,goal.y):  dx = dxL  dy = dyL  else:  dx = dxR  dy = dyR      temp = None  if d < r:  temp = (dx \* s, dy \* s)  elif r <= d and d <= s+r:  temp = (b \* (s + r -d) \* dx, b \* (s + r - d) \* dy)  elif d > s+r:  temp = (0,0)    return temp  ####################################################################  # Calculate repulsive fields on a given location.  ####################################################################  def generateRepulsiveField(x, y, obsticles):  total = [0,0]    for o in obsticles:  temp = generateAnRepulsiveField(x,y,o)  total[0] += temp[0]  total[1] += temp[1]    return total  ####################################################################  # Generate a single atractive vector.  ####################################################################  def genAnAttractiveField(x, y, goal):  r = 1.5  s = 30.0  al = 1.0/s    d = distance(x,y,goal)    theta = math.atan2(goal.y - y, goal.x - x)    temp = None  if d < r:  temp = (0.0,0.0)  elif r <= d and d <= s+r:  temp = (al\*(d-r)\*math.cos(theta), al\*(d-r)\*math.sin(theta))  elif d > s+r:  temp = (al\*s\*math.cos(theta), al\*s\*math.sin(theta))    return temp  ####################################################################  # Return the closest goal.  ####################################################################  def getMinGoal(x,y,goals):  amin = distance(x,y,goals[0])  minGoal = goals[0]    for g in goals:  temp = distance(x,y,g)  if temp < amin:  minGoal = g  amin = temp    return minGoal  ####################################################################  # Genertes the attractive vector given every possible goal.  ####################################################################  def generateAttractiveField(x, y, goals):  total = [0,0]    minGoal = getMinGoal(x,y,goals)    return genAnAttractiveField(x,y,minGoal)  ####################################################################  # Calculate a Tangential field  ####################################################################  def generateTangentialFields(x, y, obsticles, goal):  total = [0,0]    for o in obsticles:  temp = generateAnRepulsiveField(x, y, o, True, goal)  total[0] += temp[0]  total[1] += temp[1]    return total  ####################################################################  # Generate the potential field for a given point.  ####################################################################  def generatePotentialField(x,y,flags,obsticles):    tan = generateTangentialFields(x,y,obsticles, getMinGoal(x,y,flags))  att = generateAttractiveField(x,y,flags)  rep = generateRepulsiveField(x,y,obsticles)    return (tan[0] + att[0] + rep[0],  tan[1] + att[1] + rep[1])  ####################################################################  # Struct: basically a point (x,y)  ####################################################################  class HomeBaseCenter(object):  def \_\_init\_\_(self, x, y):  self.x = x  self.y = y  ####################################################################  ####################################################################  ## Agent  ####################################################################  ####################################################################  class Agent(object):  """Class handles all command and control logic for a teams tanks."""    ####################################################################  # Constructor  ####################################################################  def \_\_init\_\_(self, bzrc):  self.bzrc = bzrc  self.constants = self.bzrc.get\_constants()  self.obsticles = self.bzrc.get\_obstacles()  self.commands = []  self.error0 = 0    bases = self.bzrc.get\_bases()  for base in bases:  if base.color == self.constants['team']:  self.homeBase = base    self.homeBaseCenter = HomeBaseCenter(self.homeBase.corner1\_x +  ((self.homeBase.corner3\_x - self.homeBase.corner1\_x) / 2.0),  self.homeBase.corner1\_y +  ((self.homeBase.corner3\_y - self.homeBase.corner1\_y) / 2.0))    self.timeSet = [0,0,0,0,0,0,0,0,0,0] #For deltaTime  self.error0 = [0,0,0,0,0,0,0,0,0,0] #For deltaError    ####################################################################  ####################################################################  def tick(self, time\_diff):  mytanks, othertanks, flags, shots = self.bzrc.get\_lots\_o\_stuff()    self.mytanks = mytanks  self.othertanks = othertanks  self.flags = self.removeMyFlag(flags)  self.shots = shots  self.enemies = [tank for tank in othertanks  if tank.color != self.constants['team']]    #Clear Commands  self.commands = []    for tank in mytanks:  self.sendToCaptureFlag(tank, time\_diff)    results = self.bzrc.do\_commands(self.commands)    ####################################################################  # Determine if capturing a flag or returning it.  ####################################################################  def determinedGoals(self, tank):  if tank.flag == '-':  return self.flags  else:  return [self.homeBaseCenter]    ####################################################################  # Return the potential field to lead a tank home.  ####################################################################  def generateHomePotentialField(self,x,y):  return generatePotentialField(x,y,[self.homeBaseCenter],self.obsticles)    ####################################################################  # Perform calculations from a potential field and translate them  # into a speed and anglular velocity for a tank.  # PDControlor for Angular velocity.  ####################################################################  def sendToCaptureFlag(self, tank, time\_diff):  self.Kp = 0.60  self.Kd = 0.50    deltaPosition = generatePotentialField(tank.x, tank.y,  self.determinedGoals(tank),  self.obsticles)    newTheta = math.atan2(deltaPosition[1], deltaPosition[0])  newTheta = newTheta + 2 \* math.pi if newTheta < 0 else newTheta  posTankAngle = tank.angle + 2 \* math.pi if tank.angle < 0 else tank.angle    #Calculate the error  error = newTheta - posTankAngle  error = error - 2 \* math.pi if error > math.pi else error    #PDController  derivative = (error - self.error0[tank.index])/ (time\_diff - self.timeSet[tank.index])  newAngleVelocity = (self.Kp \* error) + (self.Kd \* derivative)    #Calculate Speed as a function of angular velocity  speed = math.sqrt(math.pow(deltaPosition[0], 2) + math.pow(deltaPosition[1], 2))  tempAngle = math.fabs(newAngleVelocity)  if tempAngle >= 1:  speed = 0.0  else:  speed = 1.0 - tempAngle    #Generate Command  captureFlagCommand = Command(tank.index, speed, newAngleVelocity, True)  self.commands.append(captureFlagCommand)    #Save error and time for derivative  self.error0[tank.index] = error  self.timeSet[tank.index] = time\_diff    return    ####################################################################  # Set command to move to given coordinates.  ####################################################################  def move\_to\_position(self, tank, target\_x, target\_y):  target\_angle = math.atan2(target\_y - tank.y,  target\_x - tank.x)  relative\_angle = self.normalize\_angle(target\_angle - tank.angle)  command = Command(tank.index, 1, 2 \* relative\_angle, True)  self.commands.append(command)    ####################################################################  # Make any angle be between +/- pi.  ####################################################################  def normalize\_angle(self, angle):  angle -= 2 \* math.pi \* int (angle / (2 \* math.pi))  if angle <= -math.pi:  angle += 2 \* math.pi  elif angle > math.pi:  angle -= 2 \* math.pi  return angle    ####################################################################  # Remove my flag from the list.  ####################################################################  def removeMyFlag(self, flags):  temp = None  for f in flags:  if f.color == self.constants['team']:  temp = f    flags.remove(temp)  return flags    ####################################################################  # Return all of the flags in the game save my own.  ####################################################################  def getTargetFlags(self):  return self.removeMyFlag(self.bzrc.get\_flags())    ####################################################################  # Make any angle be between +/- pi.  ####################################################################  def printPFields(self):  obsticles = self.bzrc.get\_obstacles()  flags = self.getTargetFlags()    #printer = PFPrinter('aFields.gpi')  #printer.printObsticles(obsticles)  #printer.printPotentialFields(lambda x,y: generateAttractiveField(x, y,flags))    #printer = PFPrinter('rFields.gpi')  #printer.printObsticles(obsticles)  #printer.printPotentialFields(lambda x,y: generateRepulsiveField(x, y, obsticles))    #printer = PFPrinter('tFields.gpi')  #printer.printObsticles(obsticles)  #printer.printPotentialFields(lambda x,y: generateTangentialFields(x, y, obsticles))    printer = PFPrinter('homeFields.gpi')  printer.printObsticles(obsticles)  printer.printPotentialFields(lambda x,y: self.generateHomePotentialField(x, y))    printer = PFPrinter('pFields.gpi')  printer.printObsticles(obsticles)  printer.printPotentialFields(lambda x,y: generatePotentialField(x, y, flags, obsticles))  def main():  # Process CLI arguments.  try:  execname, host, port = sys.argv  except ValueError:  execname = sys.argv[0]  print >>sys.stderr, '%s: incorrect number of arguments' % execname  print >>sys.stderr, 'usage: %s hostname port' % sys.argv[0]  sys.exit(-1)    # Connect.  #bzrc = BZRC(host, int(port), debug=True)  bzrc = BZRC(host, int(port))    agent = Agent(bzrc)    prev\_time = time.time()    # Run the agent  try:  while True:  time\_diff = time.time()  agent.tick(time\_diff)  except KeyboardInterrupt:  print "Exiting due to keyboard interrupt."  bzrc.close()  if \_\_name\_\_ == '\_\_main\_\_':  if len(sys.argv) == 4:  execname, host, port, printMe = sys.argv    if printMe == "-p":  bzrc = BZRC(host, int(port))  agent = Agent(bzrc)  agent.printPFields()  bzrc.close()    else:  main() |

**PFAgent Complete Code**

|  |
| --- |
| import sys |
| import math |
| import time |
| import random |
|  |
| from bzrc import BZRC, Command |
|  |
| class Agent(object): |
| """Class handles all command and control logic for a teams tanks.""" |
|  |
| def \_\_init\_\_(self, bzrc): |
| self.bzrc = bzrc |
| self.constants = self.bzrc.get\_constants() |
| self.commands = [] |
|  |
| self.counter = 0 |
| self.forwardCount = 8 |
| self.shootCount = 2 |
|  |
| def tick(self, time\_diff): |
| """Some time has passed; decide what to do next.""" |
| mytanks, othertanks, flags, shots = self.bzrc.get\_lots\_o\_stuff() |
| self.mytanks = mytanks |
| self.othertanks = othertanks |
| self.flags = flags |
| self.shots = shots |
| self.enemies = [tank for tank in othertanks if tank.color != |
| self.constants['team']] |
|  |
| self.commands = [] |
|  |
| self.counter += 1 |
|  |
| for tank in self.mytanks: |
| self.moveForwardOrTurn(tank) |
| self.randomShoot(tank) |
|  |
| results = self.bzrc.do\_commands(self.commands) |
|  |
| def moveForwardOrTurn(self, tank): |
|  |
| if self.counter < 80: |
| #Go Forward |
| command = Command(tank.index, 1.0, 0.0, False) |
| self.commands.append(command) |
| elif self.counter > 70 and self.counter < 160: |
| #Stop and Turn |
| command = Command(tank.index, 0.5, 1.0, False) |
| self.commands.append(command) |
| else: |
| #Reset |
| self.counter = 0 |
|  |
| return |
|  |
| def randomShoot(self, tank): |
| test = random.random() |
| if test < .5: |
| command = Command(tank.index, 0, 0, True) |
| self.commands.append(command) |
| return |
|  |
|  |
| def main(): |
| # Process CLI arguments. |
| try: |
| execname, host, port = sys.argv |
| except ValueError: |
| execname = sys.argv[0] |
| print >>sys.stderr, '%s: incorrect number of arguments' % execname |
| print >>sys.stderr, 'usage: %s hostname port' % sys.argv[0] |
| sys.exit(-1) |
|  |
| # Connect. |
| #bzrc = BZRC(host, int(port), debug=True) |
| bzrc = BZRC(host, int(port)) |
|  |
| agent = Agent(bzrc) |
|  |
| prev\_time = time.time() |
|  |
| # Run the agent |
| try: |
| while True: |
| time\_diff = time.time() - prev\_time |
| agent.tick(time\_diff) |
| except KeyboardInterrupt: |
| print "Exiting due to keyboard interrupt." |
| bzrc.close() |
|  |
|  |
| if \_\_name\_\_ == '\_\_main\_\_': |
| main() |