Christopher LaJon Morgan

Scott Crunkleton

CS 470: BZRFlags Tutorial

Sep 17, 2013

**Really Dumb Agent**

My really dumb agent uses a call to the function tick as a counter. Every time tick is called it treats this as if a tenth of a second has transpired. For 80 ticks ~8second the tank will simply plow forward as fast as possible. Then for another approximate 8 seconds the tank will slow to half speed and begin turning at full speed. Then the process repeats.

The whole time, however, the tank picks a random number between 0 and 1, if that number is less than .5 it fires its cannon.

~The complete source code is found at the end of this document.

**Potential Fields Implementation and Description**

* Attractive Fields (Write Up and Implementation) Why and How

The idea of an attractive field is to pull an object to a goal. My attractive fields implementation works first by finding the distance and angle that any given x, y position is from a single goal. It then returns a change in x and a change in y that will bring the given x, y position closer to the desired goal.

The magnitude of this change in x and y is governed by two parameters: *radius* and *seek*. Radius governs how big the goal (attractive) object actually is, where seek describes an outer parameter about the goal at which an object should slow down. For my implementation I chose a radius of 1.5, the idea being to kept the tanks at full throttle as long as possible until they were almost right on top of the flag. My implementation also set a seek distance of 30.0 which barely slowed the tanks down as they approached the goal.

I used attractive fields in my implementation to direct tanks to both capture an enemy’s flag and return it back to base. This code is found in the function *def genAnAttractiveField(x, y, goal)* below.

* Repulsive Fields (Write Up and Implementation) Why and How

The simple repulsive field works very similar to an attractive field except that instead of pulling an object closer it drives an object away. My implementation follows a similar process as the one used to generate attractive fields, the idea being to simply switch the signs on the changes in x and y that are generated as attractive fields. The implementation also changes the effects of *radius* and *seek*.

The meanings of radius and seek for repulsive fields change in the following ways. Radius changes from meaning stop you have arrived to get away as fast as you can. Seek in this case means the radius around the obstacle for which the repulsion is actually felt. In our game all obstacles are rectangles, so I chose my radius to be the bounding radius of each rectangle. In other words, it was the smallest radius for which the entire rectangle was contained. I also picked a fairly large seek of 60 to help keep any tanks from getting stuck on corners.

I used repulsive forces in my implementation to drive tanks away from obstacles that got in between them and their goals. This code can be found, in a modified way, in the function *def generateAnRepulsiveField(x,y, obsticle, makeItTangent=False, goal=None)* below.

* Tangential Fields (Write Up and Implementation) Why and How

A tangential field is a field that pushes at a 90 angle from a repulsive field.

I used tangential fields in my implementation to help prevent tanks from getting stuck in the world. The tangential field would guild a tank around an obstacle. Without these fields there would be a lot of holes in the field map where attractive fields and repulsive field would simply just cancel each other out.

I calculated tangential fields by calling the repulsive field function (def generateAnRepulsiveField(x,y, obsticle, makeItTangent=False, goal=None)) with the makeItTangent variable set to True. This would cause a repulsive field to be generated but with +- 90 added to the calculation angel. The plus or minus was determined by with calculation would drive the tank closer to the goal.

* Tuning Process
  + Size of Radius and Seek

These variables were simple to tweak. Simply took a guess and then printed the potential fields. When the fields looked about right that was it. However, toward the end of the project I found that there was one special case where a tank could get trapped going in a circle on the corner of an obstacle. This led me to increase the repulsive field radius as to slow the tank down before it got right in on the corner.

* + Constants b and alpha

These constants were harder to figure out. After looking at it, I came to the conclusion that they should be 1/seek. This allowed the change in x and the change in y to settle more frequently with values between 0 and 1 when x and y were within the seek field.

* + Intelligent Tangential Fields  
    Another place of entrapment for my tanks was approaching an obstacle and having it tell the tank to go the long way around the obstacle. This caused the tank to end up in an infinite circle. So I modified my tangential code to intelligently direct the tank around an obstacle in the direction closest to its goal.

\* Graphs On Next Page.

|  |  |
| --- | --- |
| Attractive Fields with Closest Goal Modification  Macintosh HD:Users:clm:Desktop:BYU-FALL-2013:CS470-AI:PotentialFieldsLab:Screenshot from 2013-09-28 16&%19&%03.png | Repulsive Fields  Macintosh HD:Users:clm:Desktop:BYU-FALL-2013:CS470-AI:PotentialFieldsLab:Screenshot from 2013-09-28 17&%19&%48.png |
| Simple Tangential Fields  Macintosh HD:Users:clm:Desktop:BYU-FALL-2013:CS470-AI:PotentialFieldsLab:Screenshot from 2013-09-28 18&%38&%53.png | Total with Goal and Tangential Modification  Macintosh HD:Users:clm:Desktop:BYU-FALL-2013:CS470-AI:PotentialFieldsLab:Screenshot from 2013-09-28 18&%46&%23.png |

* Graphs Of Potential Fields (From Red’s Perspective)

**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**

I didn’t win the first time! I got stuck in a black hole of goals. I thought simply summing all of the attractive fields to each goal would pull a tank into the closest goal. Not true. Upon actually thinking about this it was obvious this wouldn’t work. So, I modified my attractive field calculation such that a tank will only go after one goal at a time. The selection of this goal is simply the closest goal from a list of all possible goals. This did the trick.

The second mod I had to make was the return to base goal. So after getting a flag I simply change a tanks goal to the center of its base instead of the closest enemy flag.

1. **Run against two dumb agents, you should win**

The really dumb agents can get off some effective shots on occasion. I won! Not much that was new here.

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.

I also saw a major bug in my code. I noticed that when a tank was forced to go strictly in the – or direction the error would explode and cause the tank to nuts as the error bounced between and –. I had to modify the code to perform a transformation between tank coordinates and 2.

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

First of all I won! Second, one interesting observation is that the delay over the network caused my tanks to slightly overshoot their goals. The tanks had a little too much velocity and would overshoot the flag slightly every time. There seems to be a little more variance over a network versus running on localhost.

Really not much here, they run around like crazy, I capture their flag and return it.

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 because they simply shoot straight and my tanks end up on the same shortest distance path. I need a much more complicated firing algorithm.

Again, they ran around, I captured the flag and returned it.

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 a continuous space. His agents had a hard time with this calculation and would occasionally get stuck spinning in circles. Also, part of the reason that I won the game was because of the speed at which my tanks were moving. My tanks were much more effective at getting to the shortest path and then accelerating the maximum speed. I was able to capture the flag at twice his rate. In other words, speed and distance traveled were a big deal.

In other words, my tanks had a much more effective PD controller controlling angular velocity. My tanks were able to hit the desired angle faster while not overshooting the mark. Also my speed was a function of angular velocity, therefore, the only time my tanks took off at full speed was when they were facing close to the desired direction. His overshot the angular mark and his speed function was a kludge.

**Time Contributed**

Christopher Morgan -- 35 hours

Scott Crunkleton -- x hours

**PFAgent Files**

<https://github.com/chris-clm09/bzflag.git>  
In the bzagents dir.

**PFAgent 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() |

**Really Dumb Agent 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() |