Zyron Melancon

ZM1158

AI for Games Assessment

## I.0 Requirements Documentation

### I.1 Description of Problem

**Name:** Artificial Intelligence for Games

**Problem statement:** Apply artificial intelligence in game development

**Problem specification:**

* Knowledge of modern AI techniques for games
* Ability to implement pathfinding
* Skills in artificial decision-making
* Ability to implement steering behaviors

### I.2 Input Information

Program settings are set through the initialization arguments of the game class. Real-time input happens in the Pygame window with mouse button and keyboard presses.

### I.3 Output Information

Information is output through the Pygame window. If debug setting is set to true, each agent will display their position and velocity.

### I.4 User Interface Information

The Pygame window will display all UI information. The information provided is determined by the arguments given when initializing the game.

## II.0 Design Documentation

### II.1 System Architecture Description

The program is in the structure of AIE’s standard. It has a start, update, draw, and end. All initialization happens in “start”, all computation per frame happens in “update”, all visual objects are made visible during “draw”. The program loop stops and is shut down during “end”

This game has a user-settable number of agents that will immediately spawn and start wandering at the start of the program.

Each agent has functions for each basic AI steering function. Seek, Wander, and Flee. Their behavior in response to the target’s position is determined in real-time while the program is running.

Each agent is stylized as a wireframe triangle that is rotated towards its current heading. If debug mode is true, there are lines that represent the agent’s current velocity, and the last force applied, and text showing their current position, velocity, and state.

## III.0 Implementation Documentation

### III.1 Program Code

#### program.py

from game import Game

PROGRAM = Game([800,800], 60, 100, False, True)

PROGRAM.run()

#### game.py

import pygame

import random

from gametemp import GameTemplate

from veclib import \*

from seeker import \*

import constants

class Game(GameTemplate):

def \_\_init\_\_(self, screensize, frps, numseekers, debug, eyecandy):

self.\_SCREENWIDTH = screensize[0]

self.\_SCREENHEIGHT = screensize[1]

self.\_NUMSEEKERS = numseekers

self.\_SEEKERS = []

self.\_FPS = frps

self.\_CLOCK = pygame.time.Clock()

self.\_TARGET = Vec2(0, 0)

self.\_SCREEN = None

self.\_END = False

self.\_SEEKERSBRAVE = True

self.\_SEEKERSRANGE = 150

self.\_MAXSPEED = 3

self.\_ACTIVE = True

self.\_IGNORE = False

self.\_FONT = False

self.\_SEEKERSIZE = [20, 20]

self.\_DEBUG = debug

self.\_BG = True

self.\_EYECANDY = eyecandy

def \_startup(self):

pygame.init()

for i in range(self.\_NUMSEEKERS):

self.\_SEEKERS.append(Seeker(self.\_SEEKERSIZE, Vec2(random.randrange(self.\_SCREENWIDTH),random.randrange(self.\_SCREENHEIGHT)), self.\_MAXSPEED, self.\_DEBUG, self.\_EYECANDY))

self.\_SCREEN = pygame.display.set\_mode((self.\_SCREENWIDTH, self.\_SCREENHEIGHT))

pygame.display.set\_caption("Steering behavior example")

self.\_FONT = pygame.font.SysFont('Arial', 20, False, False)

self.\_TARGET = Vec2(self.\_SCREENWIDTH/2, self.\_SCREENHEIGHT/2)

def \_update(self):

for event in pygame.event.get():

if event.type == pygame.QUIT:

self.\_END = True

# If clicked, set target position

if event.type == pygame.MOUSEMOTION:

self.\_TARGET.x, self.\_TARGET.y = pygame.mouse.get\_pos()

if event.type == pygame.MOUSEBUTTONDOWN:

if pygame.mouse.get\_pressed()[0]:

self.\_SEEKERSBRAVE = not self.\_SEEKERSBRAVE

if pygame.mouse.get\_pressed()[2]:

self.\_IGNORE = not self.\_IGNORE

if event.type == pygame.KEYDOWN:

if pygame.key.get\_pressed()[pygame.K\_SPACE]:

self.\_BG = not self.\_BG

if not self.\_IGNORE:

for seeker in self.\_SEEKERS:

if magnitude(Vec2(self.\_TARGET.x - seeker.position.x, self.\_TARGET.y - seeker.position.y)) < self.\_SEEKERSRANGE:

if self.\_SEEKERSBRAVE:

seeker.applyForce(seeker.seek(self.\_TARGET),self.\_CLOCK.get\_time()/6)

else:

seeker.applyForce(seeker.flee(self.\_TARGET),self.\_CLOCK.get\_time()/6)

else:

seeker.applyForce(seeker.wander(),self.\_CLOCK.get\_time()/6)

else:

for seeker in self.\_SEEKERS:

seeker.applyForce(seeker.wander(),self.\_CLOCK.get\_time()/6)

for seeker in self.\_SEEKERS:

seeker.updatePos(self.\_CLOCK.get\_time()/6)

# Boundary stuff

if seeker.position.x <= 0:

seeker.position.x = 0

seeker.velocity.x += 1

if seeker.position.y <= 0:

seeker.position.y = 0

seeker.velocity.y += 1

if seeker.position.x >= self.\_SCREENWIDTH:

seeker.position.x = self.\_SCREENWIDTH

seeker.velocity.x -= 1

if seeker.position.y >= self.\_SCREENHEIGHT:

seeker.position.y = self.\_SCREENHEIGHT

seeker.velocity.y -= 1

if self.\_SEEKERSBRAVE == True and self.\_IGNORE == False:

self.\_rangecol = (0,255,0)

self.\_text = self.\_FONT.render('Seek', False, (255,255,255))

else:

if self.\_SEEKERSBRAVE == False and self.\_IGNORE == False:

self.\_rangecol = (255,0,0)

self.\_text = self.\_FONT.render('Flee', False, (255,255,255))

else:

self.\_rangecol = (0,0,255)

self.\_text = self.\_FONT.render('Ignore', False, (255,255,255))

def \_draw(self):

self.\_SCREEN.blit(self.\_text,(0,0))

if self.\_BG:

self.\_SCREEN.fill((0,0,0))

pygame.draw.circle(self.\_SCREEN, self.\_rangecol, (self.\_TARGET.x, self.\_TARGET.y), self.\_SEEKERSRANGE, 2)

for seeker in self.\_SEEKERS:

seeker.draw(self.\_SCREEN)

pygame.display.flip()

def run(self):

self.\_startup()

while self.\_END == False:

self.\_CLOCK.tick(self.\_FPS)

self.\_update()

self.\_draw()

pygame.quit()

#### gametemp.py

import pygame

from constants import \*

class GameTemplate(object):

def \_\_init\_\_(self):

pygame.init()

def \_startup(self):

return True

def \_update(self):

return True

def \_draw(self):

'''draw'''

def \_shutdown(self):

'''shutdown'''

#### gameobject.py

import pygame

import math

import random

from veclib import \*

class GameObject(object):

def \_\_init\_\_(self, surf, posin, maxin, debug, eyecandy):

self.\_pos = posin

self.\_vector = Vec2(0,0)

self.\_max = maxin

self.\_heading = Vec2(0,0)

self.\_surface = pygame.Surface(surf, pygame.SRCALPHA)

self.\_lastforce = Vec2(0,0)

self.\_textsurface = pygame.Surface((surf[0]\*5,surf[1]\*5), pygame.SRCALPHA)

self.\_FONT = pygame.font.SysFont('Arial', 15, False, False)

self.\_text = 'Meh'

self.\_points = ([0,0], [0,surf[1]], [surf[0],surf[1]/2], [0,0])

self.\_debug = debug

self.\_eyecandy = eyecandy

def draw(self, screen):

if self.\_eyecandy == True:

pygame.draw.lines(self.\_surface, (random.randrange(255),random.randrange(255),random.randrange(255)), True, self.\_points, 2)

else:

pygame.draw.lines(self.\_surface, (100,100,100), True, self.\_points, 2)

thisangle = math.atan2(self.\_heading.y, self.\_heading.x) \* 180 / math.pi

if thisangle < 0:

thisangle += 360

newsurface = pygame.transform.rotate(self.\_surface, -thisangle)

screen.blit(newsurface, (self.\_pos.x - 10, self.\_pos.y - 10))

if self.\_debug:

text = self.\_FONT.render(self.\_text, False, (255,255,255))

text2 = self.\_FONT.render('V: ' + str(["%.2f" % self.\_vector.x, "%.2f" % self.\_vector.y]), False, (255,255,255))

text3 = self.\_FONT.render('P: ' + str(["%.0f" % self.\_pos.x,"%.0f" % self.\_pos.y]), False, (255,255,255))

screen.blit(text, (self.\_pos.x, self.\_pos.y + 10))

screen.blit(text2, (self.\_pos.x, self.\_pos.y + 24))

screen.blit(text3, (self.\_pos.x, self.\_pos.y + 38))

pygame.draw.line(screen, (255,0,0), (int(self.\_pos.x), int(self.\_pos.y)), (int(self.\_pos.x + self.\_vector.x \* 5), int(self.\_pos.y + self.\_vector.y \* 5)), 2)

pygame.draw.line(screen, (0,255,0), (int(self.\_pos.x), int(self.\_pos.y)), (int(self.\_pos.x + self.\_lastforce.x), int(self.position.y + self.\_lastforce.y)), 2)

#### seeker.py

import math

from veclib import \*

import random

import pygame

from gameobject import GameObject

class Seeker(GameObject):

def seek(self, target):

mag = magnitude(Vec2(target.x - self.\_pos.x, target.y - self.\_pos.y))

V = normalize(Vec2(target.x - self.\_pos.x, target.y - self.\_pos.y))

MaxV = Vec2(V.x \* self.\_max / (mag/3), V.y \* self.\_max / (mag/3))

Force = Vec2(MaxV.x - (self.\_vector.x / 50), MaxV.y - (self.\_vector.y / 50))

self.lastforce = Force

self.\_text = 'Seek'

return Force

def flee(self, target):

self.\_heading = normalize(self.\_vector)

mag = magnitude(Vec2(self.\_pos.x - target.x, self.\_pos.y - target.y))

V = normalize(Vec2(self.\_pos.x - target.x, self.\_pos.y - target.y))

MaxV = Vec2(V.x \* self.\_max / (mag/3), V.y \* self.\_max / (mag/3))

Force = Vec2(MaxV.x - (self.\_vector.x / 100), MaxV.y - (self.\_vector.y / 100)) # Tried to make a smoother steer

self.lastforce = Force

self.\_text = 'Flee'

return Force

def wander(self):

self.\_timer = 0

norm = normalize(self.\_vector)

direc = random.randrange(314)

Force = normalize(Vec2(math.cos(direc), math.sin(direc)))

self.lastforce = Force

self.\_text = 'Wander'

return Vec2(Force.x/3 + self.\_heading.x, Force.y/3 + self.\_heading.y)

def applyForce(self, force, deltatime):

self.\_vector.x += force.x \* deltatime

self.\_vector.y += force.y \* deltatime

if magnitude(self.\_vector) > self.\_max:

norm = normalize(self.\_vector)

self.\_vector = Vec2(norm.x \* self.\_max, norm.y \* self.\_max)

def updatePos(self, deltatime):

self.\_pos.x += self.\_vector.x \* deltatime

self.\_pos.y += self.\_vector.y \* deltatime

self.\_heading = normalize(self.\_vector)

@property

def position(self):

return self.\_pos

@property

def velocity(self):

return self.\_vector

@position.setter

def position(self, value):

self.\_pos = value

@velocity.setter

def velocity(self, value):

self.\_vector = value

#### veclib.py

import math

class Vec2:

def \_\_init\_\_(self, xin, yin):

self.\_x = xin

self.\_y = yin

@property

def x(self):

return self.\_x

@x.setter

def x(self, value):

self.\_x = value

@property

def y(self):

return self.\_y

@y.setter

def y(self,value):

self.\_y = value

@property

def xy(self):

return [self.\_x, self.\_y]

def magnitude(vector):

return math.sqrt(vector.x\*vector.x + vector.y\*vector.y)

def normalize(vector):

mag = magnitude(vector)

if mag > 0:

return Vec2(vector.x/mag, vector.y/mag)

else:

return Vec2(0, 0)

def dot(vecone, vectwo):

return (vecone.x \* vectwo.x) + (vecone.y \* vectwo.y)

## IV.0 Operating Directions

**Mouse Position:** Target will follow your mouse cursor, always

**Left-Click:** Switch between seek and flee functions for all agents

**Right-Click:** Toggle ignore setting

**Spacebar:** Toggles screen filling (cool effects in eyecandy mode)

To edit preferences, open “program.py” in an editor. Preferences are set in the initialization of Game at line 3.

Game([window size], FPS, number of agents, debug on/off, eyecandy on/off)

**Currently known bugs:**

The agents will seem to jerk and turn in random directions while it’s on top of the target point, even though they are seemingly keeping their position

**Dependencies:**

-Latest version of Pygame

-Python 2.7.14