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

Real-time input happens in the Pygame window with keyboard presses. Key functions are explained in **IV.0**

### I.3 Output Information

Information is output through the Pygame window. Each square represents a node. If the node is red, it is “unwalkable”. Otherwise, the node will display their own F, tentative G, and H values.

### I.4 User Interface Information

The Pygame window will display all UI information and take keyboard input.

## 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 “startup”, all computation per frame happens in “update”, all visual objects are made visible during “draw”. The program loop stops and is shut down during “shutdown”

The program starts with a graph of nodes, drawn as a grid of squares on the screen. All nodes start off as “walkable” at the start of the program. The A-Star algorithm searches for a path of nodes that would be the quickest way from a start to a destination. A-Star will also try to find paths around unwalkable nodes to reach its destination.

Using the arrow keys, you are able to move the node selector, which is indicated by a white circle. Specified keys will trigger functions that can set the states of the selected node. Pressing spacebar will start or stop the A-Star animation.

While the A-Star animation is running, lines will be drawn to indicate the nodes that it has checked. The lines represent which nodes are parented and in the open list. If the algorithm finds a path, it will display a green line representing the least-costly path from the start to the destination. If the destination is not found due to the destination being in a space that is impossible to reach, it will continue to check around until there are no nodes in the open list.

## III.0 Implementation Documentation

### III.1 Program Code

#### program.py

from astarlive import AStarGame

Game = AStarGame()

Game.\_startup()

#### astarlive.py

# Import a library of functions called 'pygame'

import pygame

import drawablenode

from drawablenode import \*

from math import pi

import random

from constants import \*

from gametemp import GameTemplate

class AStarGame(GameTemplate):

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

# Initialize the game engine

pygame.init()

# Set the height and width of the SCREEN

self.SCREEN = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

# Create the nodes

self.NODES = {}

for i in range(ROWS):

for j in range(COLS):

self.NODES[str([i,j])] = DrawableNode([i,j])

# Add adjacents to each node

for n in self.NODES:

# Right

if str([self.NODES[n].posx + 1,self.NODES[n].posy]) in self.NODES:

self.NODES[n].adjacents.append(self.NODES[str([self.NODES[n].posx + 1,self.NODES[n].posy])])

# Up-Right

if str([self.NODES[n].posx + 1,self.NODES[n].posy + 1]) in self.NODES:

self.NODES[n].adjacents.append(self.NODES[str([self.NODES[n].posx + 1,self.NODES[n].posy + 1])])

# Up

if str([self.NODES[n].posx, self.NODES[n].posy + 1]) in self.NODES:

self.NODES[n].adjacents.append(self.NODES[str([self.NODES[n].posx, self.NODES[n].posy + 1])])

# Up-Left

if str([self.NODES[n].posx - 1,self.NODES[n].posy + 1]) in self.NODES:

self.NODES[n].adjacents.append(self.NODES[str([self.NODES[n].posx - 1,self.NODES[n].posy + 1])])

# Left

if str([self.NODES[n].posx - 1,self.NODES[n].posy]) in self.NODES:

self.NODES[n].adjacents.append(self.NODES[str([self.NODES[n].posx - 1,self.NODES[n].posy])])

# Down-Left

if str([self.NODES[n].posx - 1,self.NODES[n].posy - 1]) in self.NODES:

self.NODES[n].adjacents.append(self.NODES[str([self.NODES[n].posx - 1,self.NODES[n].posy - 1])])

# Down

if str([self.NODES[n].posx,self.NODES[n].posy - 1]) in self.NODES:

self.NODES[n].adjacents.append(self.NODES[str([self.NODES[n].posx,self.NODES[n].posy - 1])])

# Down-Right

if str([self.NODES[n].posx + 1,self.NODES[n].posy - 1]) in self.NODES:

self.NODES[n].adjacents.append(self.NODES[str([self.NODES[n].posx + 1,self.NODES[n].posy - 1])])

# Window description

pygame.display.set\_caption("Example code for A-Star")

# Loop until the user clicks the close button.

self.DONE = False

self.CLOCK = pygame.time.Clock()

pygame.font.init()

self.font1 = pygame.font.Font(None, 14)

self.font2 = pygame.font.Font(None, 28)

# Set the destination and self.start point here

self.start = self.NODES["[0, 0]"]

self.dest = self.NODES[str([ROWS-1,COLS-1])]

self.selnode = self.NODES["[0, 0]"]

self.currentNode = self.start

self.opnlist = []

self.clslist = []

self.currentNode.g = 1

self.currentNode.h = (abs((self.dest.posx - self.currentNode.posx)) + abs((self.dest.posy - self.currentNode.posy))) \* 10

self.currentNode.f = self.currentNode.h + self.currentNode.g

self.opnlist.append(self.currentNode)

self.keepgoing = False

def \_update(self):

# Leave this out and we will use all CPU we can.

self.CLOCK.tick(FPS)

for event in pygame.event.get(): # User did something

if event.type == pygame.QUIT: # If user clicked close

self.DONE = True # Flag that we are self.DONE so we exit this loop

if self.keepgoing == True:

# Do the A-Star thing if safe

if self.dest.walkable == True and self.start.walkable == True:

if self.opnlist and self.currentNode != self.dest:

self.currentNode = self.opnlist[0]

#Add the current node to the closed list

self.opnlist.remove(self.currentNode)

self.clslist.append(self.currentNode)

#Go through adjacents and calculate G, H, F

for adj in self.currentNode.adjacents:

if adj not in self.clslist and adj.walkable == True:

if adj not in self.opnlist:

self.opnlist.append(adj)

adj.g = 10 + self.currentNode.g

# Manhattan (absolute distance from current node to destination)

adj.h = (abs((self.dest.posx - adj.posx)) + abs((self.dest.posy - adj.posy))) \* 10

adj.f = adj.h + adj.g

if adj.parent:

if adj.parent.g > self.currentNode.g:

adj.parent = self.currentNode

else:

adj.parent = self.currentNode

# Sort open list by value of f

self.opnlist.sort(key=lambda n: n.f)

# If pressed uparrow, move destination up

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

if self.selnode.posy > 0: self.selnode = self.NODES[str([self.selnode.posx,self.selnode.posy-1])]

# If pressed downarrow, move destination down

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

if self.selnode.posy < COLS-1: self.selnode = self.NODES[str([self.selnode.posx,self.selnode.posy+1])]

# If pressed leftarrow, move destination left

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

if self.selnode.posx > 0: self.selnode = self.NODES[str([self.selnode.posx-1,self.selnode.posy])]

# If pressed rightarrow, move destination right

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

if self.selnode.posx < ROWS-1: self.selnode = self.NODES[str([self.selnode.posx+1,self.selnode.posy])]

# If pressed S, set self.start to selected

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

self.start = self.selnode

self.currentNode = self.start

self.opnlist = []

self.clslist = []

self.opnlist.append(self.currentNode)

# Reset all node values

for noodles in self.NODES:

self.NODES[noodles].parent = None

self.NODES[noodles].h = 0

self.NODES[noodles].f = 0

self.NODES[noodles].g = 0

self.currentNode.g = 1

self.currentNode.h = abs((self.dest.posx - self.currentNode.posx)) + abs((self.dest.posy - self.currentNode.posy))

self.currentNode.f = self.currentNode.h + self.currentNode.g

# If pressed D, set destination to selected

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

self.dest = self.selnode

self.start = self.currentNode

self.opnlist = []

self.clslist = []

self.opnlist.append(self.currentNode)

# Reset all node values

for noodles in self.NODES:

self.NODES[noodles].parent = None

self.NODES[noodles].h = 0

self.NODES[noodles].f = 0

self.NODES[noodles].g = 0

self.currentNode.g = 1

self.currentNode.h = abs((self.dest.posx - self.currentNode.posx)) + abs((self.dest.posy - self.currentNode.posy))

self.currentNode.f = self.currentNode.h + self.currentNode.g

# If pressed A, make selected walkable true/false

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

if self.selnode.walkable == True: self.selnode.walkable = False

else: self.selnode.walkable = True

# If pressed C, clear all unwalkable nodes

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

for noodles in self.NODES:

self.NODES[noodles].walkable = True

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

for noodles in self.NODES:

if self.NODES[noodles] != self.start and self.NODES[noodles] != self.dest:

self.NODES[noodles].walkable = not self.NODES[noodles].walkable

else:

self.NODES[noodles].walkable = True

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

self.keepgoing = not self.keepgoing

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

for noodles in self.NODES:

self.NODES[noodles].walkable = random.choice([True,False])

self.start.walkable = True

self.dest.walkable = True

def \_draw(self):

# Clear the self.SCREEN and set the self.SCREEN background

self.SCREEN.fill(DARK)

# Draw the nodes

for o in self.NODES:

self.NODES[o].draw(self.SCREEN, self.font1)

# Draw the self.start, destination, and selected nodes as circles

pygame.draw.circle(self.SCREEN, GREEN, (self.start.screenpos[0] + WIDTH/2, self.start.screenpos[1] + HEIGHT/2), 6)

pygame.draw.circle(self.SCREEN, REDRED, (self.dest.screenpos[0] + WIDTH/2, self.dest.screenpos[1] + HEIGHT/2), 6)

pygame.draw.circle(self.SCREEN, WHITE, (self.selnode.screenpos[0] + WIDTH/2, self.selnode.screenpos[1] + HEIGHT/2), 6)

pygame.draw.circle(self.SCREEN, REDRED, (self.currentNode.screenpos[0] + WIDTH/2, self.currentNode.screenpos[1] + HEIGHT/2), 6)

# Draw guess lines

if EYECANDY:

for o in self.NODES:

if self.NODES[o].parent:

pygame.draw.line(self.SCREEN, (random.randrange(100, 255), random.randrange(100, 255), random.randrange(100, 255)), (self.NODES[o].screenpos[0] + WIDTH/2, self.NODES[o].screenpos[1] + HEIGHT/2),

(self.NODES[o].parent.screenpos[0] + WIDTH/2,

self.NODES[o].parent.screenpos[1] + HEIGHT/2), 4)

else:

for o in self.NODES:

if self.NODES[o].parent:

pygame.draw.line(self.SCREEN, RED, (self.NODES[o].screenpos[0] + WIDTH/2, self.NODES[o].screenpos[1] + HEIGHT/2),

(self.NODES[o].parent.screenpos[0] + WIDTH/2,

self.NODES[o].parent.screenpos[1] + HEIGHT/2), 4)

# Draw quickest path lines

if self.dest.parent:

currn = self.dest

while currn != self.start:

pygame.draw.line(self.SCREEN, GREEN, (currn.screenpos[0] + WIDTH/2, currn.screenpos[1] + HEIGHT/2), (currn.parent.screenpos[0] + WIDTH/2, currn.parent.screenpos[1] + HEIGHT/2), 6)

currn = currn.parent

# Go ahead and update the self.SCREEN with what we've drawn.

# This MUST happen after all the other drawing commands.

bg = pygame.Surface((self.SCREEN.get\_size()[0] / 3, self.SCREEN.get\_size()[1] / 3))

bg.fill(BLACK)

textrect = bg.get\_rect()

pygame.display.flip()

def \_shutdown(self):

# Be IDLE friendly

pygame.quit()

def \_startup(self):

while self.DONE == False:

self.\_update()

self.\_draw()

self.\_shutdown()

#### constants.py

# Visual settings

EYECANDY = False

FPS = 15

# Window and Node sizes

PAD = (5, 5)

ROWS = 50

COLS = 25

WIDTH = 30

HEIGHT = 30

SCREEN\_WIDTH = ROWS \* (PAD[0] + WIDTH) + PAD[1]

SCREEN\_HEIGHT = COLS \* (PAD[0] + HEIGHT) + PAD[1]

# Define the colors we will use in RGB format

BLACK = (0, 0, 0)

WHITE = (255, 255, 255)

DARK = (15, 15, 15)

BLUE = (0, 0, 100)

GREEN = (0, 180, 0)

RED = (100, 0, 100) # Actually purple

REDRED = (255, 0, 0)

#### drawablenode.py

import pygame

import math

class DrawableNode(object):

'''drawable node'''

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

# astar vars

self.posx = inpos[0]

self.posy = inpos[1]

self.adjacents = []

self.parent = None

self.\_walkable = True

self.\_g = 0

self.\_h = 0

self.\_f = 0

# drawing vars

SIZE = 30

self.width = SIZE

self.height = SIZE

self.id = id

self.index = (self.posx, self.posy)

self.x = (5 + self.width) \* self.posx + 5

self.y = (5 + self.height) \* self.posy + 5

self.pos = (self.width \* self.posx, self.height \* self.posy)

self.screenpos = (self.x, self.y)

self.rect = pygame.Rect(self.x, self.y, self.width, self.height)

self.surface = pygame.Surface((self.width, self.height))

self.dirty = False

self.\_color = (15, 15, 15)

# properties

@property

def walkable(self):

return self.\_walkable

@walkable.setter

def walkable(self, value):

blue = (0, 0, 255)

red = (255, 0, 0)

self.\_walkable = value

# if it's set to walkable change to white

# this will mark it as undirty

if value:

self.color = (15, 15, 15)

else:

self.color = (100, 0, 0)

@property

def f(self):

return self.\_f

@property

def g(self):

return self.\_g

@property

def h(self):

return self.\_h

@f.setter

def f(self, value):

self.\_f = value

@g.setter

def g(self, value):

self.\_g = value

self.\_f = self.\_g + self.\_h

@h.setter

def h(self, value):

self.\_h = value

self.\_f = self.\_g + self.\_h

@property

def color(self):

return self.\_color

@color.setter

# manual setting of colors will mark them dirty so they will stay

def color(self, value):

white = (255, 255, 255)

red = (100, 0, 0)

if value is red:

self.\_color = value

self.dirty = True

else:

self.\_color = value

self.\_color = value

def info(self):

print("pos = ", self.pos)

ids = ""

for i in self.adjacents:

ids += " " + str(i.id)

print("neighbors:", ids)

print("index: ", self.index)

def draw(self, screen, font, init=True, text=True):

# pygame.draw.rect(screen, self.\_color, self.rect)

self.surface.fill(self.\_color)

screen.blit(self.surface, self.screenpos)

if self.walkable:

# create some text to go on the fill

# info to display

# render the text

textf = font.render("F= " + str(self.f), True, (50, 50, 50))

textg = font.render("G= " + str(self.g) +

"H= " + str(self.h), True, (50, 50, 50))

# set it's position/parent

textfpos = (self.x, self.y) # top left

textgpos = (self.x, self.y + self.height - 10) # bot left

# center it

# draw the square

if init and text:

screen.blit(textf, textfpos)

screen.blit(textg, textgpos)

#### 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'''

## IV.0 Operating Directions

**Arrow Keys:** Move the node selector

**A:** Toggle selected node’s walkable state

**S:** Set selected node to start

**D:** Set selected node to destination

**Z:** Randomize the walkable state of all nodes in the graph

**X:** Invert the walkable state of all nodes in the graph

**C:** Sets all unwalkable nodes to walkable

**Space:** Toggle A-Star animation

**Currently known bugs:**

If you set nodes to unwalkable on a path that’s already been checked, A-Star does not go back to re-correct itself. Instead, it will draw a line through the unwalkable node, using the path it already took. The same applies if you invert or randomize the nodes during the animation.