Brett Stelly

March 28, 2018

Assessment Four

1. **Requirements Documentation**
   1. **Description of Problem**

**Name:** A\* Unit Test

**Problem Statement:** Implement the A\* Algorithm by writing a program that runs the algorithm and correctly finds the path. Also make an application that visually shows the A\* Algorithm working

**Problem Specifications**: The algorithm should find the correct shortest path from start to goal

* 1. **Input information**

The user interacts with the application by using the Enter/Return, Ctrl, Shift, C, and Esc keys, and the mouse

* 1. **Output Information**

The application displays the grid that the user can interact with. When the user interacts with the graph, the application shows any changes made. Once the A\* algorithm is initiated, the application shows the algorithm in action by drawing it to the screen.

* 1. **User Interface**

The user can click and drag the start and goal squares to place them on the grid. The squares in the grid can also be clicked on to turn them into a wall.

1. **System Architecture**

**Node.py**

Prototype: def \_\_init\_\_(self, pos)

Arguments: A Vector2

Description: The initializer for creating an instance of the Node Class

Precondition: None

Postcondition: An instance of the Node class is created

Prototype: def calculate\_g\_score(self, other)

Arguments: An instance of the Node class

Description: Calculates the G-Score for a node

Precondition: There must be two instances of the Node class

Postcondition: The g\_score for the node is calculated

Prototype: def calculate\_h\_score(self, other)

Arguments: An instance of the Node class

Description: Calculates the H-Score for a node

Precondition: There must be two instances of the Node class

Postcondition: The nodes H-Score is calculated

Prototype: def calculate\_f\_score(self)

Arguments: None

Description: Calculates the F-Score for a node

Precondition: There must be an instance of the Node class

Postcondition: The nodes F-Score is calculated

Prototype: def set\_parent(self, other)

Arguments: An instance of the node class

Description: Sets the parent of a node to the node that is passed in

Precondition: There must be two instances of the Node class

Postcondition: The node's parent is set to the node passed in

Prototype: def toggle\_state(self)

Arguments: None

Description: toggles the traversable state of a node

Precondition: There must be an instance of the Node class

Postcondition: The nodes traversability is assigned the opposite of its current state

Prototype: def get\_x(self)

Arguments: None

Description: returns the node position's x-value

Precondition: There must be an instance of the Node class

Postcondition: The x-value is returned

Prototype: def get\_y(self)

Arguments: None

Description: returns the node position's y-value

Precondition: There must be an instance of the Node class

Postcondition: The y-value is returned

**Graph.py**

Prototype: def \_\_init\_\_(self, length, height)

Arguments: An interger for the length of the graph and an interger for the height of the graph

Description: Creates an instance of the Graph class and populates it with nodes

Precondition: None

Postcondition: An instance of the Graph class is created

Prototype: def create\_nodes(self)

Arguments: None

Description: Creates the nodes in the Graph and assigns then a position

Precondition: There must be an instance of the Graph class

Postcondition: A graph is created with nodes

Prototype: def get\_neighbors(self, current\_node)

Arguments: An instance of the node class

Description: Finds valid neighbor positions and returns them in a list

Precondition: There must be an instance of the Graph class

Postcondition: A list of nodes with neighboring postitions is returned

Prototype: def \_\_getitem\_\_(self, index)

Arguments: An interger

Description: returns the node in the index that corresponds to the passed in interger

Precondition: There must be an instance of the graph class already created

Postcondition: A node is returned

**a\_star.py**

Prototype: def \_\_init\_\_(self, graph, start, end)

Arguments: An instance of the graph class and two Node objects for the start and goal

Description: Creates an instance of the AStar class

Precondition: There must be an instance of the Graph class and two instances of the Node class

Postcondition: An instance of the AStar class is created

Prototype: def set\_start(self, start\_node)

Arguments: A Node object

Description: Sets the start node of the graph to the node passed in

Precondition: There must be an instance of the AStar class and a node object

Postcondition: The AStar instance is given a new starting node

Prototype: def set\_goal(self, goal\_node)

Arguments: A Node object

Description: Sets the goal ode of the graph to the node passed in

Precondition: There must be an instance of the AStar class and a node object

Postcondition: The AStar instance is given a new goal node

Prototype: def find\_current(self)

Arguments: None

Description: Finds the current node in the path

Precondition: There must be an instance of the AStar class

Postcondition: The current node is set and moved to the closed list and out of the open list

Prototype: def find\_path(self)

Arguments: None

Description: Generates a path from start to goal node

Precondition: The closed list has to contain the goal node

Postcondition: A list of nodes is returned in order from start to goal node

Prototype: def reset(self)

Arguments: None

Description: Resets the AStar algorithm for multiple runs

Precondition: There must be an instance of the AStar algorithm

Postcondition: The AStar instance has and empty open list, closed list, and path. All node

parents are set to none

Prototype: def update(self, start\_node, goal\_node)

Arguments: A Node object for the starting node and a Node object for the goal node

Description: Updates the start and goal node and runs the AStar algorithm

Precondition: There must be an instance of the Astar algorithm and two Node objects

Postcondition: The instance of Astar is updated with a path

**draw\_utils.py**

**class Rectangle:**

Prototype: def \_\_init\_\_(self, draw\_surface, color, position, scale\_x, scale\_y)

Arguments: A draw surface, color, Vector2, an interger for length, and interger for height

Description: Creates a Rectangle object and draws a rectangle to the screen

Precondition: There must be a drawing surface created using pygame

Postcondition: A Rectangle object is created and is drawn to the screen

Prototype: def draw(self)

Arguments: None

Description: Draws a rectangle to the screen

Precondition: There must be a drawing surface created using pygame

Postcondition: A rectangle is drawn to the screen

**Class Circle**

Prototype: def \_\_init\_\_(self, draw\_surface, color, position, radius)

Arguments: A draw surface, color, Vector2, and an interger for radius

Description: Creates a circle object and draws a circle to the screen

Precondition: There must be a drawing surface created using pygame

Postcondition: A Circle object is created and drawn to the screen

Prototype: def draw(self)

Arguments: None

Description: Draws a Circle to the screen

Precondition: There must be a drawing surface created using pygame

Postcondition: A Circle is drawn to the screen

**Class Line**

Prototype: def \_\_init\_\_(self, draw\_surface, color, start\_pos, end\_pos, width)

Arguments: A drawing surface, color, a Vector2 for start pos, a Vector2 for end pos,

and an int for width

Description: Creates an instance of the Line class and draws a line to the screen

Precondition: There must be a drawing surface created using pygame

Postcondition: An instance of the Line class is created and a line is drawn to the screen

Prototype: def draw(self)

Arguments: None

Description: Draws a line to the screen

Precondition: A drawing surface must be created using pygame

Postcondition: A line is drawn to the screen

**Class Ellipse**

Prototype: def \_\_init\_\_ (self, draw\_surface, color, position, scale\_x, scale\_y)

Arguments: a draw surface, color, Vector2, an int for length and an int for height

Description: Creates an instance of the Ellipse class and draws an ellipse to the screen

Precondition: There must be a draw surface created using pygame

Postcondition: An Ellipse object is created and an ellipse is drawn to the screen

Prototype: def draw(self)

Arguments: None

Description: Draws an ellipse to the screen

Precondition: There must be a draw surface created using pygame

Postcondition: An ellipse is drawn to the screen

**Class Text**

Prototype: def \_\_init\_\_(self, text, font\_theme, font\_size, color, draw\_surface, x\_pos, y\_pos)

Arguments: A string, A string for font name, an int for font size, color, draw surface, int

for x pos, and int for y pos

Description: Creates a text object and draws it to the screen

Precondition: A draw surface must be created using pygame

Postcondition: A text object is created and drawn to the screen

Prototype: def draw(self)

Arguments: None

Description: Draws text to the screen

Precondition: There must be a draw surface created using pygame

Postcondition: Text is drawn to the screen

**draw\_astar\_visuals**

Prototype: def \_\_init\_\_(self, node, color, draw\_pos, scale\_x, scale\_y, draw\_surface)

Arguments: a Node object, a color, a Vector2, an int for length, an int for height,

and a draw surface

Description: Creates an instance of the NodeVisual class and displays a rectangle

Precondition: None

Postcondition: An instance of the NodeVisual class is created

Prototype: def \_\_init\_\_(self, astar, node\_offset, draw\_surface)

Arguments: an instance of astar, an int for node offset, and a draw surface

Description: Creates an instance of the GraphVisual class

Precondition: There must be an instance of AStar already created and a draw surface created

using pygame

Postcondition: An instance of the GraphVisual class is created

Prototype: def gen\_visual\_nodes(self)

Arguments: None

Description: Generates the visual nodes and gives them colliders

Precondition: There must be an instance of GraphVisual

Postcondition: An instance of GraphVisual has a grid of visual nodes

Prototype: def draw\_nodes(self)

Arguments: None

Description: Draws the visual nodes every frame

Precondition: There must be an instance of GraphVisual

Postcondition: The screen displays the visual nodes

Prototype: def draw\_path(self)

Arguments: None

Description: Draws the path returned from AStar using the visual nodes

Precondition: There must be an instance of GraphVisual and an instance of Astar

must retun a path

Postcondition: The screen now displays a line that travels from start to goal

Prototype: def draw\_text(self)

Arguments: None

Description: Draws text to the screen

Precondition: There must be an instance of GraphVisual

Postcondition: Text is displayed to the screen

Prototype: def clear\_grid(self)

Arguments: None Clears the grid by resetting astar, and setting traversability of all

nodes to True

Description: Clears the grid by resetting astar, and setting traversability of all

nodes to True

Precondition: There must be an instance of GraphVisual

Postcondition: The grid is reset to a default state

Prototype: def clear\_screen(self)

Arguments: None

Description: Wipes the screen to the background color and redraws text

Precondition: There must be a draw surface created using pygame

Postcondition: The screen is wiped to the background color

Prototype: def sort\_visual\_nodes\_in\_closed\_list(self)

Arguments: None

Description: Sorts closed list nodes for GraphVisual in the same order as the astar closed

list

Precondition: There must be an instance of GraphVisual

Postcondition: The close\_list\_nodes are sorted in the same order as the astar closed list

Prototype: def draw\_closed\_list(self)

Arguments: None

Description: Draws an animated closed list to the screen

Precondition: There must be an instance of GraphVisual with A\* algorithm finished running

Postcondition: The closed list is drawn to the screen

Prototype: def draw\_parents(self)

Arguments: None

Description: Draws the parents for every node in the open or closed list

Precondition: There must be an instance of GraphVisual with A\* algorithm finished running

Postcondition: The screen has lines drawn from each node to its parent

Prototype: def draw\_node\_information(self)

Arguments: None

Description: Displays the G, H, and F score of every node by drawing them using text on every

visual node in the closed or open list

Precondition: There must be an instance of GraphVisual with A\* finished running

Postcondition: Every visual node in the closed or open list now displays its own F, G, and H

score

Prototype: def update(self, event)

Arguments: Pygame Event

Description: Updates the application according to the Pygame event passed in

Precondition: There must be an instance of GraphVisual and a pygame event must be passed in

Postcondition: The application is updated

**application.py**

Prototype: def \_\_init\_\_(self, width, height)

Arguments: an int for width and and int for height

Description: Creates an instance of the Application class

Precondition: None

Postcondition: An instance of the Application class is created

Prototype: def update(self)

Arguments: None

Description: Updates the application by passing in a pygame event to the GraphVisual update

Precondition: None

Postcondition: The application is updated

1. **Read Me**

In order to run the application, you must double click “main.py”. Once you have the application running, you should see the graph and to the right, you should see the instructions. To find a path, click and drag the start and goal node onto the grid. You can click on the empty grid squares to turn them into a wall. You can also click and drag over them. When you are ready to start the pathfinding, press the Enter/Return key. You will then see the algorithm in action trying to find the shortest path to the goal. Once the goal is found, a yellow line indicating the path will be drawn from start to finish. You can press the Shift key to display each nodes parent. The Ctrl key can be pressed to display each nodes F, G, and H scores. If you want to quit at any time, you can press the Escape key, or you can exit out of the application window.