Tina the Toll Troll

By Timothy Couch

Written in Processing

To download and run the full program to see that everything works as intended, please visit the following website:

<https://www.dropbox.com/s/x7vo7c3wds0hlzo/sketch_CS330TinaPathfinding.zip?dl=1>

sketch\_CS330TinaPathfinding.pde

//sketch\_CS330TinaPathfinding - runs the game, adds stuff to the map, etc.

//Timothy Couch

/\*\*

\* Change this to enable/disable grid drawing and stuff

\*\*/

**boolean** debug = **true**;

/\*\*

\* This variable controls the demo instance

\* 0 - basic path, 1 - path to harry, 2 - path around obstacles

\*\*/

**int** variation = 0;

**boolean** gameStart = **false**;

Grid grid = **new** Grid(29, 38, 16);

Pather pather = **new** Pather();

Objects objects = **new** Objects();

Keys keys = **new** Keys();

Tina tina;

Player player;

Harry harry;

PVector safeSpace = **new** PVector(19, 5);

PVector bridge = **new** PVector(5, 31);

**void** settings()

{

size(grid.gridWidth \* grid.gridSize, grid.gridHeight \* grid.gridSize);

}

**void** setup()

{

}

**void** StartGame()

{

gameStart = **true**;

player = (Player) objects.addGrid(**new** Player(**new** PVector(17, 31)));

tina = (Tina) objects.addGrid(**new** Tina(bridge));

**if** (variation == 1)

harry = (Harry) objects.addGrid(**new** Harry(**new** PVector(19, 8)));

**if** (variation == 2)

{

objects.addGrid(**new** Block(**new** PVector(12, 4)));

objects.addGrid(**new** Block(**new** PVector(12, 5)));

objects.addGrid(**new** Block(**new** PVector(12, 6)));

objects.addGrid(**new** Block(**new** PVector(12, 7)));

objects.addGrid(**new** Block(**new** PVector(12, 8)));

objects.addGrid(**new** Block(**new** PVector(12, 9)));

objects.addGrid(**new** Block(**new** PVector(12, 10)));

objects.addGrid(**new** Block(**new** PVector(13, 11)));

objects.addGrid(**new** Block(**new** PVector(7, 15)));

objects.addGrid(**new** Block(**new** PVector(8, 15)));

objects.addGrid(**new** Block(**new** PVector(9, 15)));

objects.addGrid(**new** Block(**new** PVector(10, 15)));

objects.addGrid(**new** Block(**new** PVector(11, 15)));

objects.addGrid(**new** Block(**new** PVector(12, 15)));

objects.addGrid(**new** Block(**new** PVector(13, 15)));

objects.addGrid(**new** Block(**new** PVector(14, 15)));

objects.addGrid(**new** Block(**new** PVector(15, 16)));

objects.addGrid(**new** Block(**new** PVector(15, 17)));

objects.addGrid(**new** Block(**new** PVector(16, 18)));

objects.addGrid(**new** Block(**new** PVector(17, 19)));

objects.addGrid(**new** Block(**new** PVector(17, 20)));

objects.addGrid(**new** Block(**new** PVector(17, 10)));

objects.addGrid(**new** Block(**new** PVector(18, 11)));

objects.addGrid(**new** Block(**new** PVector(19, 12)));

objects.addGrid(**new** Block(**new** PVector(20, 12)));

objects.addGrid(**new** Block(**new** PVector(21, 12)));

}

}

**void** draw()

{

objects.KeysDown(keys);

objects.draw();

**if** (!gameStart)

{

pushStyle();

textAlign(CENTER);

**if** (mouseY < height / 3)

fill(100);

**else** fill(0);

text("Variation 1", width / 2, height / 4);

**if** (mouseY >= height / 3 && mouseY < height \* 2 / 3)

fill(100);

**else** fill(0);

text("Variation 2", width / 2, height \* 2 / 4);

**if** (mouseY >= height \* 2 / 3)

fill(100);

**else** fill(0);

text("Variation 3", width / 2, height \* 3 / 4);

popStyle();

}

}

**void** mousePressed()

{

**if** (!gameStart)

{

**if** (mouseY < height / 3)

variation = 0;

**if** (mouseY >= height / 3 && mouseY < height \* 2 / 3)

variation = 1;

**if** (mouseY >= height \* 2 / 3)

variation = 2;

StartGame();

}

}

**void** keyPressed()

{

**if** (keys.KeyPressed(key))

{

objects.KeyPressed(key);

}

}

**void** keyReleased()

{

**if** (keys.KeyReleased(key))

{

objects.KeyReleased(key);

}

}

Keys.pde

//Keys - handles key presses and keys held down

//Timothy Couch

**class** Keys

{

ArrayList<String> keys = **new** ArrayList<String>();

**boolean** KeyPressed(**char** key)

{

**boolean** downPrev = IsKeyDown(key);

**if** (!IsKeyDown(key))

keys.add(str(key).toUpperCase());

**return** (IsKeyDown(key) != downPrev);

}

**boolean** KeyReleased(**char** key)

{

**boolean** downPrev = IsKeyDown(key);

**if** (IsKeyDown(key))

keys.remove(str(key).toUpperCase());

**return** (IsKeyDown(key) != downPrev);

}

**boolean** IsKeyDown(**char** key)

{

**if**(keys.indexOf(str(key).toUpperCase()) >= 0)

**return** **true**;

**return** **false**;

}

**void** ObjectKeysDown(Object o)

{

**for** (String key : keys)

o.KeyDown(key.charAt(0));

}

}

Grid.pde

//Grid - handles grid spaces, movement, and drawing

//Timothy Couch

**class** Grid

{

**int** gridWidth;

**int** gridHeight;

GridObject[][] gridContents;

**int** gridSize;

**public** Grid()

{

gridWidth = 16;

gridHeight = 16;

InitDefault();

}

**public** Grid(**int** gW, **int** gH)

{

**this**();

gridWidth = gW;

gridHeight = gH;

InitDefault();

}

**public** Grid(**int** gW, **int** gH, **int** gS)

{

**this**(gW, gH);

gridSize = gS;

}

**void** InitDefault()

{

**float** squareSizeX = width / gridWidth;

**float** squareSizeY = height / gridHeight;

gridSize = round(min(squareSizeX, squareSizeY));

gridContents = **new** GridObject[gridWidth][gridHeight];

**for** (**int** i = 0; i < gridWidth; i++)

**for** (**int** j = 0; j < gridHeight; j++)

gridContents[i][j] = **null**;

}

GridObject add(GridObject o)

{

gridContents[round(o.position.x)][round(o.position.y)] = o;

**return** o;

}

PVector remove(GridObject o)

{

**for** (**int** i = 0; i < gridWidth; i++)

**for** (**int** j = 0; j < gridHeight; j++)

**if** (gridContents[i][j] == o)

{

gridContents[i][j] = **null**;

**return** **new** PVector(i, j);

}

**return** **null**;

}

GridObject updateGridObject(GridObject o)

{

remove(o);

add(o);

**return** o;

}

**boolean** spaceOpen(PVector v)

{

**if** (round(v.x) >= 0 && round(v.x) < gridWidth && round(v.y) >= 0 && round(v.y) < gridHeight)

**return** gridContents[round(v.x)][round(v.y)] == **null**;

**return** **false**;

}

**void** draw()

{

pushStyle();

fill(#ffffff, 128);

noStroke();

//tint(255, 100);

**for** (**int** i = 0; i < gridWidth; i++)

**for** (**int** j = 0; j < gridHeight; j++)

**if** (gridContents[i][j] != **null**)

rect(i \* gridSize, j \* gridSize, gridSize, gridSize);

fill(#00b3b2, 128);

rect(safeSpace.x \* gridSize, safeSpace.y \* gridSize, gridSize, gridSize);

fill(#d96c00, 128);

rect(bridge.x \* gridSize, bridge.y \* gridSize, gridSize, gridSize);

popStyle();

fill(0);

**for** (**int** i = 0; i < grid.gridWidth; i++)

{

**int** x = i \* grid.gridSize;

line(x, 0, x, height);

}

**for** (**int** j = 0; j < grid.gridHeight; j++)

{

**int** y = j \* grid.gridSize;

line(0, y, width, y);

}

}

}

GridDir.pde

//GridDir - handles directions

//Timothy Couch

**static** **class** GridDir

{

**static** **int** NULL = -1;

**static** **int** UP = 0;

**static** **int** DOWN = 1;

**static** **int** LEFT = 2;

**static** **int** RIGHT = 3;

**static** **int** KeyDir(**char** key)

{

**switch** (key)

{

**case** 'W':

**return** UP;

**case** 'S':

**return** DOWN;

**case** 'A':

**return** LEFT;

**case** 'D':

**return** RIGHT;

**default**:

**return** NULL;

}

}

**static** PVector Move(**int** dir)

{

**if** (dir == GridDir.UP)

**return** **new** PVector(0, -1);

**if** (dir == GridDir.DOWN)

**return** **new** PVector(0, 1);

**if** (dir == GridDir.LEFT)

**return** **new** PVector(-1, 0);

**if** (dir == GridDir.RIGHT)

**return** **new** PVector(1, 0);

**return** **new** PVector(0, 0);

}

**static** **int** VectorDir(PVector v)

{

**int** dir = -1;

**if** (abs(v.x) >= abs(v.y))

{

**if** (v.x >= 0)

dir = GridDir.RIGHT;

**else** dir = GridDir.LEFT;

}

**else**

**if** (v.y < 0)

dir = GridDir.UP;

**else** dir = GridDir.DOWN;

**return** dir;

}

}

Objects.pde

//Objects - handles all the object actions and drawings

//Timothy Couch

**class** Objects

{

ArrayList<Object> objects = **new** ArrayList<Object>();

Object add(Object o)

{

objects.add(o);

**return** o;

}

GridObject addGrid(GridObject o)

{

add(o);

**return** grid.add(o);

}

Object remove(**int** i)

{

Object o = objects.remove(i);

**return** o;

}

Object remove(Object o)

{

**return** objects.remove(objects.indexOf(o));

}

PVector removeGrid(GridObject o)

{

remove(o);

**return** grid.remove(o);

}

**void** step()

{

**for** (Object o : objects)

{

o.step();

}

}

**void** KeysDown(Keys keys)

{

**for** (Object o : objects)

{

keys.ObjectKeysDown(o);

}

}

**void** KeyPressed(**char** key)

{

**char** k = str(key).toUpperCase().charAt(0);

**for** (Object o : objects)

{

o.KeyPressed(k);

}

}

**void** KeyReleased(**char** key)

{

**char** k = str(key).toUpperCase().charAt(0);

**for** (Object o : objects)

{

o.KeyReleased(k);

}

}

**void** draw()

{

step();

background(#73D84C);

**if** (gameStart && debug)

grid.draw();

drawObjects();

}

**void** drawObjects()

{

**for** (Object o : objects)

{

o.drawObj();

}

**for** (Object o : objects)

{

o.drawGUI();

}

}

}

Object.pde

//Object - basic object with actions, translation, and stuff

//Timothy Couch

**class** Object

{

PVector position;

**float** imageAngle;//in degrees

**float** scaleX;

**float** scaleY;

color c;//color

Object()

{

position = **new** PVector(0, 0);

imageAngle = 0;

scaleX = 1;

scaleY = 1;

c = color(0);

}

Object(PVector pos)

{

**this**();

**this**.position = pos.copy();

c = color(200);

}

**void** step()

{

}

**void** KeyPressed(**char** key)

{

}

**void** KeyReleased(**char** key)

{

}

**void** KeyDown(**char** key)

{

}

**void** drawObj()

{

pushMatrix();

translate(position.x, position.y);

rotate(radians(imageAngle));

scale(scaleX, scaleY);

pushStyle();

draw();

popStyle();

popMatrix();

}

**void** drawGUI()

{

}

**void** draw()

{

fill(c);

rect(-5, -5, 10, 10);

}

}

GridObject.pde

//GridObject - Object that conforms to the grid

//Timothy Couch

**class** GridObject **extends** Object

{

GridObject()

{

**super**();

}

GridObject(PVector position)

{

**super**(position);

}

**void** drawObj()

{

pushMatrix();

translate(position.x \* grid.gridSize, position.y \* grid.gridSize);

rotate(radians(imageAngle));

scale(scaleX, scaleY);

pushStyle();

draw();

popStyle();

popMatrix();

}

}

Block.pde

//Block - a simple path obstructor

//Timothy Couch

**class** Block **extends** GridObject

{

Block()

{

**super**();

InitDefault();

}

Block(PVector position)

{

**super**(position);

InitDefault();

}

//replacement for default constructor

**void** InitDefault()

{

c = color(50);

}

**void** draw()

{

fill(c);

noStroke();

rect(0, 0, grid.gridSize, grid.gridSize);

}

}

Movable.pde

//Movable - GridObject that has moving functionality

//Timothy Couch

**class** Movable **extends** GridObject

{

Movable()

{

**super**();

InitDefault();

}

Movable(PVector position)

{

**super**(position);

InitDefault();

}

**void** InitDefault()

{

c = color(#00d9a3);

}

**void** step()

{

**super**.step();

}

**boolean** CanMove(**int** dir, **int** spaces)

{

**if** (dir == GridDir.UP)

{

**if** (grid.spaceOpen(**new** PVector(position.x, position.y - spaces)))

**return** **true**;

}

**else** **if** (dir == GridDir.DOWN)

{

**if** (grid.spaceOpen(**new** PVector(position.x, position.y + spaces)))

**return** **true**;

}

**else** **if** (dir == GridDir.LEFT)

{

**if** (grid.spaceOpen(**new** PVector(position.x - spaces, position.y)))

**return** **true**;

}

**else** **if** (dir == GridDir.RIGHT)

{

**if** (grid.spaceOpen(**new** PVector(position.x + spaces, position.y)))

**return** **true**;

}

**return** **false**;

}

**boolean** CanMove(**int** dir)

{

**return** CanMove(dir, 1);

}

**boolean** Move(**int** dir, **int** spaces)

{

**boolean** move = **false**;

**if** (dir == GridDir.UP)

{

**if** (grid.spaceOpen(**new** PVector(position.x, position.y - spaces)))

{

position = **new** PVector(position.x, position.y - spaces);

// position.y -= spaces;

move = **true**;

}

}

**else** **if** (dir == GridDir.DOWN)

{

**if** (grid.spaceOpen(**new** PVector(position.x, position.y + spaces)))

{

position = **new** PVector(position.x, position.y + spaces);

// position.y += spaces;

move = **true**;

}

}

**else** **if** (dir == GridDir.LEFT)

{

**if** (grid.spaceOpen(**new** PVector(position.x - spaces, position.y)))

{

position = **new** PVector(position.x - spaces, position.y);

// position.x -= spaces;

move = **true**;

}

}

**else** **if** (dir == GridDir.RIGHT)

{

**if** (grid.spaceOpen(**new** PVector(position.x + spaces, position.y)))

{

position = **new** PVector(position.x + spaces, position.y);

// position.x += spaces;

move = **true**;

}

}

**if** (move)

grid.updateGridObject(**this**);

**return** move;

}

**boolean** Move(**int** dir)

{

**return** Move(dir, 1);

}

**void** draw()

{

fill(c);

ellipse(grid.gridSize / 2, grid.gridSize / 2, round(grid.gridSize \* 5 / 8), round(grid.gridSize \* 5 / 8));

}

}

Player.pde

//Player - The object that moves around (Using WASD)

//Timothy Couch

**class** Player **extends** Movable

{

**int** stepTimeCap = 5;

**int** stepTime = stepTimeCap;

Player()

{

**super**();

InitDefault();

}

Player(PVector position)

{

**super**(position);

InitDefault();

}

**void** InitDefault()

{

c = color(#d93600);

}

**void** KeyPressed(**char** key)

{

**super**.KeyPressed(key);

stepTime = stepTimeCap;

Move(GridDir.KeyDir(key));

}

**void** KeyDown(**char** key)

{

**super**.KeyDown(key);

stepTime--;

**if** (stepTime <= 0)

{

stepTime = stepTimeCap;

Move(GridDir.KeyDir(key));

}

}

}

Harry.pde

//Harry - Harry the troll who runs around in a circle in the second demo

//Timothy Couch

**class** Harry **extends** Movable

{

**int**[] pArray = {GridDir.LEFT,

GridDir.LEFT,

GridDir.UP,

GridDir.LEFT,

GridDir.UP,

GridDir.UP,

GridDir.UP,

GridDir.UP,

GridDir.RIGHT,

GridDir.UP,

GridDir.RIGHT,

GridDir.RIGHT,

GridDir.RIGHT,

GridDir.RIGHT,

GridDir.DOWN,

GridDir.RIGHT,

GridDir.DOWN,

GridDir.DOWN,

GridDir.DOWN,

GridDir.DOWN,

GridDir.LEFT,

GridDir.DOWN,

GridDir.LEFT,

GridDir.LEFT};

Path path;

**int** stepTimeCap = 5;

**int** stepTime = stepTimeCap;

**boolean** stopped = **false**;

Harry()

{

**super**();

InitDefault();

}

Harry(PVector position)

{

**super**(position);

InitDefault();

}

**void** InitDefault()

{

path = **new** Path(pArray);

c = color(#59b300);

}

**void** step()

{

**if** (**new** PVector(tina.position.x - position.x, tina.position.y - position.y).mag() < 1.5)

stopped = **true**;

**else** stopped = **false**;

**if** (!stopped)

{

stepTime--;

**if** (stepTime <= 0)

{

stepTime = stepTimeCap;

Move(path.stepLoop());

}

}

}

**void** draw()

{

**super**.draw();

**if** (debug)

path.draw();

}

/\*\*

\* This method is a simple replacement for animation. I could simply call some animator's method to animate, but that is too pseudocode-y.

\* A good pseudocode replacement would be animate();

\*/

**void** drawGUI()

{

String printString = "";

**if** (stopped)

printString = "Healing!";

fill(0);

text(printString, (position.x + 5 / 2) \* grid.gridSize - 1, (position.y + 3 / 2) \* grid.gridSize - 1);

}

}

Tina.pde

//Tina - TINAAAA!!!!!!!

//Timothy Couch

**class** Tina **extends** Movable

{

Path path;

PVector dest;

**boolean** moved = **false**;

**boolean** fighting = **false**;

**boolean** fleeing = **false**;

**boolean** healing = **false**;

**boolean** resting = **false**;

**int** energyCap = 600;

**int** energy = energyCap;

**boolean** tired = **false**;

**int** harryDist = 10;

**int** stepTimeCap = 5;

**int** stepTime = stepTimeCap;

**int** pathUpdateTimeCap = 60;

**int** pathUpdateTime = pathUpdateTimeCap;

Tina()

{

**super**();

InitDefault();

}

Tina(PVector position)

{

**super**(position);

InitDefault();

}

Tina(PVector position, **int** var)

{

**super**(position);

InitDefault();

}

**void** InitDefault()

{

dest = calcDestination();

path = MakePath();

c = color(#73ffdc);

}

PVector calcDestination()

{

**if** (!tired)

**return** bridge;

**else**

**switch** (variation)

{

**case** 0:

**return** safeSpace;

**case** 1:

**return** harry.position;

**case** 2:

**return** safeSpace;

**default**:

**return** position;

}

}

**void** step()

{

stepTime--;

**if** (stepTime <= 0)

{

stepTime = stepTimeCap;

moved = Move(path.step());

}

**if** (tired)

c = color(#004040);

**else** c = color(#73ffdc);

fighting = **false**;

fleeing = **false**;

healing = **false**;

resting = **false**;

**if** (!tired)

{

**if** (!moved)

{

**if** ((position.x != dest.x || position.y != dest.y) && finished)

path = MakePath();

energy--;

**if** (**new** PVector(player.position.x - position.x, player.position.y - position.y).mag() < 4)

{

energy -= 3;

fighting = **true**;

}

}

**if** (energy <= 0)

{

tired = **true**;

dest = calcDestination();

path = MakePath();

}

}

**else**

{

**if** (!moved)

{

**if** ((position.x != dest.x || position.y != dest.y) && finished && (variation != 1 || **new** PVector(harry.position.x - position.x, harry.position.y - position.y).mag() > harryDist))

path = MakePath();

**if** (**new** PVector(dest.x - position.x, dest.y - position.y).mag() <= 1.5)

{

**if** (variation != 1)

{

energy += 3;

resting = **true**;

}

**else**

{

energy += 4;

healing = **true**;

}

}

}

**else** fleeing = **true**;

**if** (energy >= energyCap)

{

tired = **false**;

dest = calcDestination();

path = MakePath();

}

}

pathUpdateTime--;

**if** (pathUpdateTime <= 0 && (variation != 1 || **new** PVector(harry.position.x - position.x, harry.position.y - position.y).mag() > harryDist))

{

**if** (variation == 1)

dest = calcDestination();

path = MakePath();

}

}

Path MakePath()

{

pathUpdateTime = pathUpdateTimeCap;

Path p = pather.GeneratePath(position, dest);

**return** p;

}

**void** draw()

{

**super**.draw();

**if** (debug)

path.draw();

}

/\*\*

\* This method is a simple replacement for animation. I could simply call some animator's method to animate, but that is too pseudocode-y.

\* A good pseudocode replacement would be animate();

\*/

**void** drawGUI()

{

String printString = "";

**if** (fighting)

printString = "Combat!";

**else** **if** (fleeing)

printString = "Fleeing!";

**else** **if** (resting)

printString = "Resting!";

**else** **if** (healing)

printString = "Healing!";

fill(0);

text(printString, (position.x + 5 / 2) \* grid.gridSize - 1, (position.y + 3 / 2) \* grid.gridSize - 1);

}

}

Pather.pde

//Pather - creates a path

//Timothy Couch

**class** Pather

{

Path GeneratePath(PVector source, PVector dest)

{

**if** (source.x == dest.x && source.y == dest.y)

{

**int**[] path = **new** **int**[1];

path[0] = GridDir.NULL;

**return** **new** Path(path);

}

**return** GenerateAStarPath(source, dest);

}

Path GenerateAStarPath(PVector source, PVector dest)

{

ArrayList<Node> openList = **new** ArrayList<Node>();

ArrayList<Node> closedList = **new** ArrayList<Node>();

openList.add(**new** Node(source, 0, dest, **null**));

**int** numSteps = 1;

**boolean** foundPath = **false**;

**while** (openList.size() > 0)

{

Node lowNode = openList.get(0);

**if** (lowNode.position.x == dest.x && lowNode.position.y == dest.y)

{

foundPath = **true**;

**break**;

}

**else**

{

openList.remove(lowNode);

closedList.add(lowNode);

ArrayList<PVector> adjacentList = **new** ArrayList<PVector>();

PVector adjPos = **new** PVector(lowNode.position.x - 1, lowNode.position.y);

**if** (grid.spaceOpen(adjPos) || (adjPos.x == dest.x && adjPos.y == dest.y))

adjacentList.add(adjPos);

adjPos = **new** PVector(lowNode.position.x + 1, lowNode.position.y);

**if** (grid.spaceOpen(adjPos) || (adjPos.x == dest.x && adjPos.y == dest.y))

adjacentList.add(adjPos);

adjPos = **new** PVector(lowNode.position.x, lowNode.position.y - 1);

**if** (grid.spaceOpen(adjPos) || (adjPos.x == dest.x && adjPos.y == dest.y))

adjacentList.add(adjPos);

adjPos = **new** PVector(lowNode.position.x, lowNode.position.y + 1);

**if** (grid.spaceOpen(adjPos) || (adjPos.x == dest.x && adjPos.y == dest.y))

adjacentList.add(adjPos);

**for** (PVector p : adjacentList)

{

**boolean** isOn = **false**;

**for** (Node n : openList)

**if** (n.HasSameVector(p))

{

isOn = **true**;

**break**;

}

**if** (!isOn)

**for** (Node n : closedList)

**if** (n.HasSameVector(p))

{

isOn = **true**;

**break**;

}

**if** (!isOn)

{

Node newNode = **new** Node(p, numSteps, dest, lowNode);

**boolean** added = **false**;

**for** (**int** i = 0; i < openList.size(); i++)

{

**if** (newNode.dist < openList.get(i).dist)

{

openList.add(i, newNode);

added = **true**;

**break**;

}

}

**if** (!added)

openList.add(newNode);

}

}

}

numSteps++;

}

**if** (foundPath)

{

IntList pathList = **new** IntList();

Node currentNode = openList.get(0);

**while** (currentNode.previousNode != **null**)

{

pathList.append(GridDir.VectorDir(currentNode.position.sub(currentNode.previousNode.position)));

currentNode = currentNode.previousNode;

}

pathList.reverse();

**return** **new** Path(pathList.array());

}

**return** **new** Path();

}

Path GenerateLinePath(PVector source, PVector dest)

{

PVector pos = source.copy();

IntList dirs = **new** IntList();

**int** times = 0;

**while** (pos.x != dest.x || pos.y != dest.y)

{

PVector between = **new** PVector(dest.x - pos.x, dest.y - pos.y);

**int** dir = -1;

**if** (abs(between.x) >= abs(between.y))

{

**if** (between.x >= 0)

dir = GridDir.RIGHT;

**else** dir = GridDir.LEFT;

}

**else**

**if** (between.y < 0)

dir = GridDir.UP;

**else** dir = GridDir.DOWN;

dirs.append(dir);

pos.add(GridDir.Move(dir));

times++;

**if** (times > 200)

**break**;

}

**return** **new** Path(dirs.array());

}

}

Path.pde

//Path - holds path instructions

//Timothy Couch

**class** Path

{

**int**[] path;

**int** step;

Path()

{

path = **new** **int**[1];

path[0] = GridDir.NULL;

/\* path = new int[4];

path[0] = GridDir.UP;

path[1] = GridDir.RIGHT;

path[2] = GridDir.DOWN;

path[3] = GridDir.LEFT;

\*/

step = 0;

}

Path(**int**[] p)

{

**this**();

path = p;

}

**int** step()

{

**if** (path.length > 0 && step < path.length)

{

**int** currStep = step;

step++;

**return** path[currStep];

}

**return** GridDir.NULL;

}

**int** stepLoop()

{

**if** (path.length > 0)

{

**int** currStep = step;

step++;

step %= path.length;

**return** path[currStep];

}

**return** GridDir.NULL;

}

**boolean** finished()

{

**return** step == path.length;

}

**void** draw()

{

PVector currPos = **new** PVector(0, 0);

**for** (**int** i = step; i < path.length; i++)

{

PVector prevPos = currPos.copy();

currPos.add(GridDir.Move(path[i]));

drawSegment(prevPos, currPos, i);

}

}

**void** drawSegment(PVector p1, PVector p2, **int** currStep)

{

stroke(#ffffff, round(((**float**) path.length - currStep) / (path.length - step) \* 255));

line((p1.x + .5) \* grid.gridSize, (p1.y + .5) \* grid.gridSize, (p2.x + .5) \* grid.gridSize, (p2.y + .5) \* grid.gridSize);

}

}

Node.pde

//Node - holds a vector and a score (distance from destination)

//Timothy Couch

**class** Node

{

PVector position;

**float** dist;

Node previousNode = **null**;

Node()

{

position = **new** PVector(0, 0);

dist = 0f;

}

Node(PVector p)

{

position = p;

dist = 0f;

}

Node(PVector p, **int** numSteps, PVector dest, Node prevNode)

{

position = p;

dist = numSteps + CalculateDist(dest);

previousNode = prevNode;

}

**float** CalculateDist(PVector dest)

{

**return** position.dist(dest);

}

**boolean** HasSameVector(PVector v)

{

**return** (position.x == v.x && position.y == v.y);

}

}