**Assignment #4 Report**

CPSC 481 - Artificial Intelligence

**A) Team Members: OwlHackers**

|  |  |  |  |
| --- | --- | --- | --- |
| **Names** | **Email** | **Percentage** | |
| Rocio Salguero | [salgueroroci@csu.fullerton.edu](mailto:salguero.roci@csu.fullerton.edu) | 20% | |
| Thao Nguyen | [tnguyen29@csu.fullerton.edu](mailto:tnguyen29@csu.fullerton.edu) | 20% | |
| Sharayu Shetty | [sharayu333@csu.fullerton.edu](mailto:sharayu333@csu.fullerton.edu) | [20%](mailto:sharayu333@csu.fullerton.edu) | |
| Long Nguyen | [long261vn@csu.fullerton.edu](mailto:long261vn@csu.fullerton.edu) | 20% | |
| Sanika Deshpande | sanikadesh198@csu.fullerton.edu | 20% | |

* **Project Information:**

**Project Name : hw4**

**File Names :** hw4.cpp, hw4test.cpp, package.xml, CMakeList.txt, readme.txt

**B) Strategy (or heuristic function):**

We will add machine learning to our movement function (**Steepest-ascent/descent Hill-climbing algorithm**) to predict where T-turtles will be and move to that location instead.

* **Machine learning:**

For turtles T1 and T2 that only use distance factor:

if the turtle has moved

Move to the location

Else

Find speed information

Generalize the X or Y field that isn't used.

Predict the next point using distance = time \* speed

new x or y = x or y + distance;

* **Using the same movement functions:**

**/\*------Movement Functions-------\*/**

A function that will keep moving the turtle till it catches all target turtles and uses the heuristic functions

Checks what target turtle (TX) is closest to mainturtle.

f(x) to find the heuristic value of next move - use f\_func(x,y)

To move to location - use move(speed, distance) and relative\_rotate(x,y)

**/\*-----Heuristic Functions------\*/**

double f\_func()

return g\_func() + h\_func(location); // f(n) = g(n) + h(n)

double g\_func()

return distance traveled;

double h\_func(location)

if( out of bounds or near villain turtle ){

return MAX\_VAL;

Else

return distance between main turtle and closest target turtle.

**/\*-----Evaluate function------\*/**

Test all location nodes near using f\_func() then return the location with lowest heuristic value

**C) Pseudocode:**

// Calculates the heuristic value of the next location   
// Gets the location and the target turtle to go to

double f\_func(x, y , turtletarget) {   
 return g\_func() + h\_func(x, y, turtletarget);   
}   
  
// Returns the distance traveled

double g\_func() {   
 return turtleDist;  
}  
   
**// Returns MAX\_VAL if near villain or out of bounds, else returns the distance to   
// nearest target turtle**double h\_func(x, y, turtletarget)

{ if (out\_of\_bounds() or near\_villain() )  
 return MAX\_VAL;   
 if(turtle == 0 || turtle == 1)

{

// only distance factor is used → use generalization for these two turtles

// check if moved

if (!moved)

{

// the turtle is stationary → go to where it is

return distFunc(nextX, nextY, targets[turtle].x, targets[turtle].y);

}

// if the turtle is moving, use generalization method

time\_t curr\_time;

//find its speed

tinfo[turtle].speedx = (x2 - x1)/(t2 - t1);

tinfo[turtle].speedy = (y2 - y1)/(t2 - t1);

// generalize speed almost 0 ergo ignore that direction

double dist; double next;

if(generalize x values)

{

//predict next y value

predictedY = targets[turtle].y + dist;

return distFunc(nextX, nextY, targets[turtle].x, predictedY);

}

else if(generalize y values)

{

//predict next y value

predictedX = targets[turtle].x + dist;

return distFunc(nextX, nextY, predictedX, targets[turtle].y);

}

}

else //turtle T3

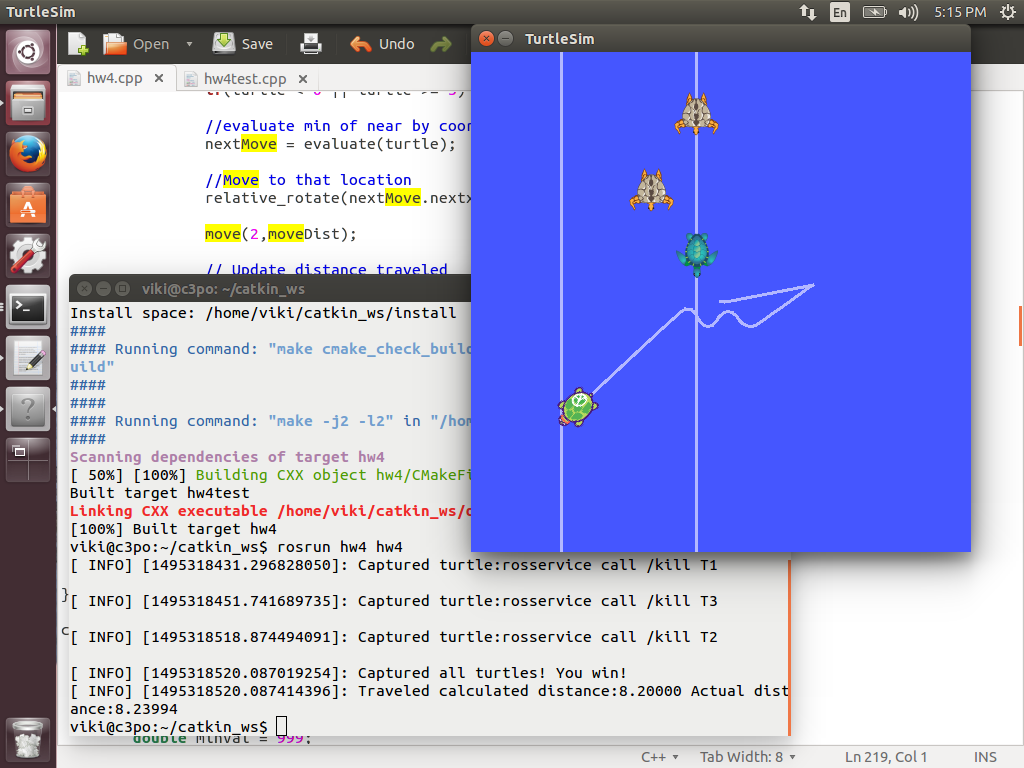
{

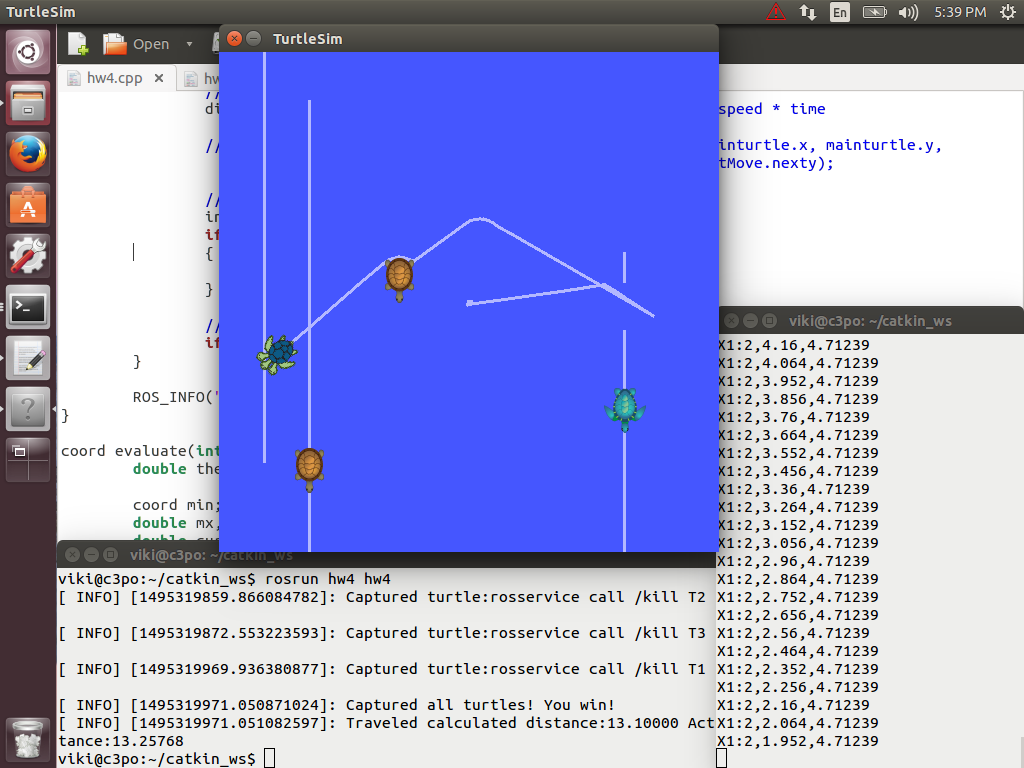
//have to keep track of turning, default heuristic for now

return distFunc(nextX, nextY, targets[turtle].x, targets[turtle].y);

}   
}   
**// Main function**main() {  
 // Spawn target turtles   
 // Spawn villain turtles  
 // Calls moveAI() function to start moving the mainturtle  
}   
  
// Calculates the closest target turtle  
int closest\_target() {  
 closest = -1;  
 double dist = 999, double current;   
 for i = 0 to target\_turtle\_number   
 current = distance (mainturtle.x, mainturtle.y , turtle[i].x , turtle[i].y)  
 if( current < dist)   
 dist = current;   
 closest = turtle[i];  
 return closest;   
}  
  
// Moves mainturtle until all 3 target turtles are captured or mainturtle is captured by Villain   
void moveAI() {  
 nextMove;   
 turtle = closest\_target();   
 while(true) {  
 turtle = closest\_target();   
 //evaluate min of near by   
 nextMove = evaluate(turtle);   
 //Move to that location  
 relative\_rotate(nextMove.nextx, nextMove.nexty);   
 move(0.5,moveDist);  
  
 // Update distance traveled  
 distanceTraveled += 0.5 \* (moveDist/0.5); // distance += speed \* time  
 //check if target captured   
 if(all targets captured) break;   
 //check if failed   
 else if( check\_fail() ) {  
 ROS\_INFO("Main turtle killed. Failed!");  
 system("rosservice call /kill mainTurtle");  
 return;   
 }   
 }   
 // Output that the target turtles are captured  
 // Output the total distance traveled  
}  
// Evaluates the nearby nodes and returns the location with the least f\_func()   
// (or heuristic) value.   
coord evaluate(int currtarget) {  
 //get the minimum f\_func() from the point near mainturtle  
 for location = 0 to 2\*PI   
 current = f\_func(location.x, location.y) ;   
 if(current < minVal) {  
 minVal = current; minLocation = location;   
 return minLocation;   
}

# **D) Screenshot of A Successful Run:**





# **E) References:**

<http://www.site.uottawa.ca/~ppayeur/SMART/PAPERS/IETrans1995.pdf>

<https://page.mi.fu-berlin.de/rojas/neural/chapter/K4.pdf>