**Discord Chat:** [**https://discord.gg/aBeXebZ**](https://discord.gg/aBeXebZ)

You can name your project name and cpp file just have to edit launch, cmake, and package.xml

Project Name:

File Name:

Launch File:

========================================================

**Q&A:**

<https://docs.google.com/document/d/1r_tbYITKfYjE6N6beaX55hPPdO28yW3jk0KdI79RY78/edit>

* **T1 will move back and forth in a fixed boundary**. For this turtle you need to focus on "distance" factor (attribute) to identify its behavior pattern. For this turtle, Rote learning to remember the fixed boundary can be an approach you can use.
* **T2 will move back and forth as well but not necessarily in a fixed boundary**. However, if you observe the "distance" factor, you will notice that T2 moves based on certain pattern. For T2, think about using generalization or induction.
* **T3 will move by two factors "distance" and "rotation".** So your program needs to find out the move pattern by two attributes "distance" and "rotation angle" using a learning method like decision tree or simple probability calculation (see the rock-paper-scissor example in lecture).
* Remember that the main objective of this assignment is to use any learning method (discussed in class such as Rote learning, generalization or induction, and supervised learning such as decision tree or perceptron learning) to learn the T turtle move patterns and capture all T turtles.
* You don't have to worry about finding out X turtles' move pattern. **For X turtles, just try to avoid being captured by X turtles**. Note that one of X turtles may chase after turtle1 when turtle1 stays on the same location for too long time.

========================================================

**catkin\_create\_pkg hw4**

Create **src** folder: Modify CMake and Project.xml. Add hw4test.cpp (from Titanium) and hw4.cpp (our file)

Navigate to folder and **catkin\_make** to compile

Source you build: **source devel/setup.bash**

**Without launch file:**

**roscore**

**rosrun turtlesim turtlesim\_node**

**rosrun hw4 hw4test**

**rosrun hw4 hw4**

//Add launch file to main folder

//To launch: **roslaunch turtlecatch .launch**

========================================================

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 2.8.3)

project(hw4)

## Find catkin macros and libraries

find\_package(catkin REQUIRED COMPONENTS roscpp geometry\_msgs)

catkin\_package()

## Specify additional locations of header files

## Your package locations should be listed before other locations

include\_directories(include ${catkin\_INCLUDE\_DIRS})

## Declare a cpp executable

add\_executable(hw4 hw4.cpp)

add\_executable(hw4test hw4test.cpp)

## Specify libraries to link a library or executable target against

target\_link\_libraries(hw4test ${catkin\_LIBRARIES})

target\_link\_libraries(hw4 ${catkin\_LIBRARIES})

========================================================

**package.xml**

<?xml version="1.0"?>

<package>

<name>hw4</name>

<version>0.0.1</version>

<description>CPSC 481 Assignment 4</description>

<maintainer email="tnguyen29@csu.fullerton.edu">Thao Nguyen</maintainer>

<maintainer email="salgueroroci@csu.fullerton.edu">Rocio Salguero</maintainer>

<maintainer email="sharayu333@csu.fullerton.edu">Sharayu Shetty</maintainer>

<maintainer email="long261vn@csu.fullerton.edu">Long Nguyen</maintainer>

<maintainer email="sanikadesh198@csu.fullerton.edu">Sanika Deshpande</maintainer>

<license>BSD</license>

<buildtool\_depend>catkin</buildtool\_depend>

<build\_depend>geometry\_msgs</build\_depend>

<run\_depend>geometry\_msgs</run\_depend>

<build\_depend>turtlesim</build\_depend>

<run\_depend>turtlesim</run\_depend>

<build\_depend>roscpp</build\_depend>

<run\_depend>roscpp</run\_depend>

</package>

========================================================

**X kills Main, Main kills T.**

X turtles : face down. 1 of them (X3) not moving, others go straight up and down (movePattern1)

T turtles : face up. 1 of them (T3) runs in special pattern , other 2 go straight up and down

**3 different Patterns:**

distance = delta\_time\*speed ; delta\_time = t1 - t0; t1 = ros::Time::now().toSec();

**movePattern1:** just moving with a distance, if hit the wall then run backward.

**movePattern2:** distance = distance \* ((loopCnt % 2)+1); look like he will improve this one later to chase us.

**movePattern3:** moveDistance, setDesiredOrientation and then moveDistance, setDesiredOrientation (with desired\_angle\_radians can be changed depend in “rdir”)

**The hw4test.cpp now:**

**T1, X1, X2 = movePattern1 ; T2 = movePattern2; T3 = movePattern3**

movePattern1(HW::tturtles[0], \_tpubs[0], speed, distance); **//T1**

movePattern1(HW::xturtles[0], \_xpubs[0], speed, distance); **//X1**

movePattern1(HW::xturtles[1], \_xpubs[1], speed, distance); **//X2**

movePattern2(HW::tturtles[1], \_tpubs[1], speed, distance, loopCnt, HW::xturtles[2], \_xpubs[2]); **//T2**

movePattern3(HW::tturtles[2], \_tpubs[2], speed, distance, rdir); **//T3**

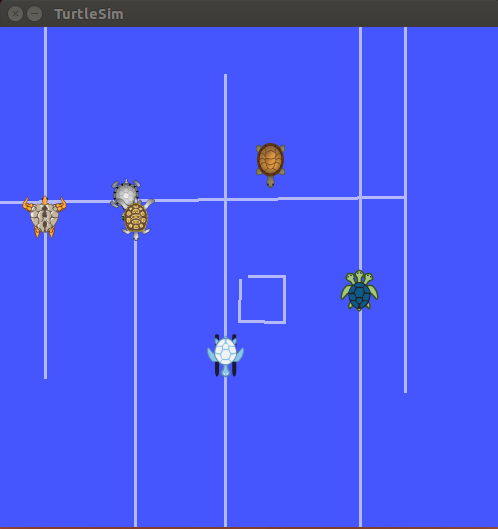
Run 1: T3 runs in square shape



Run 2: it kills our main turtle….!



Run 3: they kill me again… too dangerous !



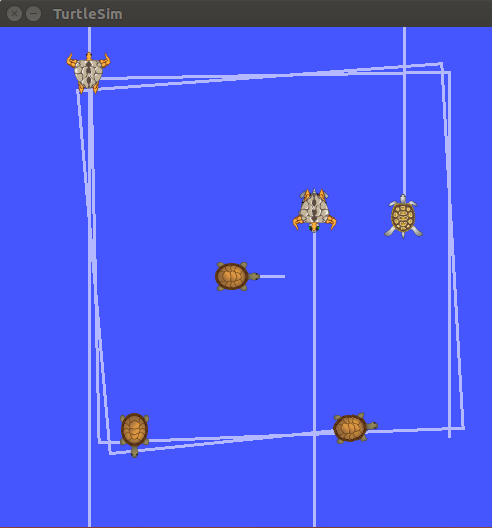
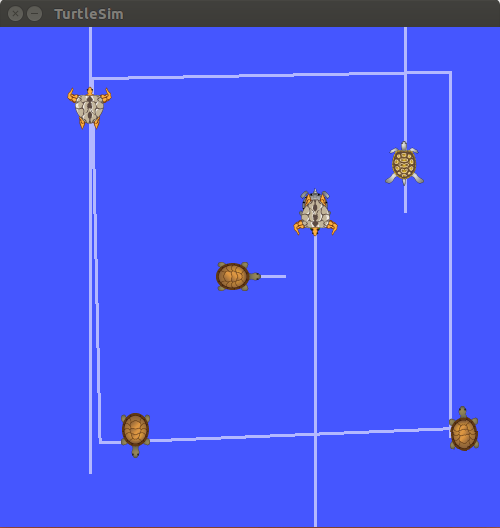
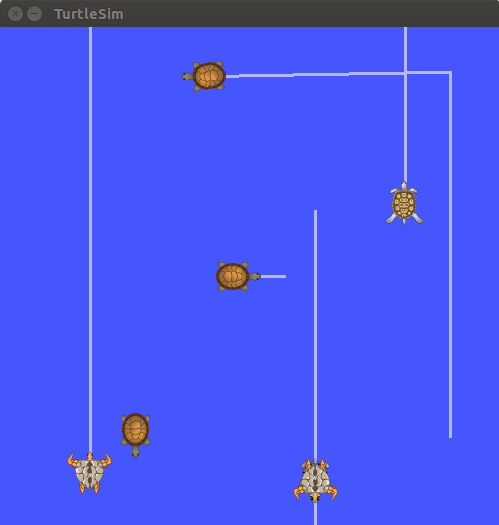
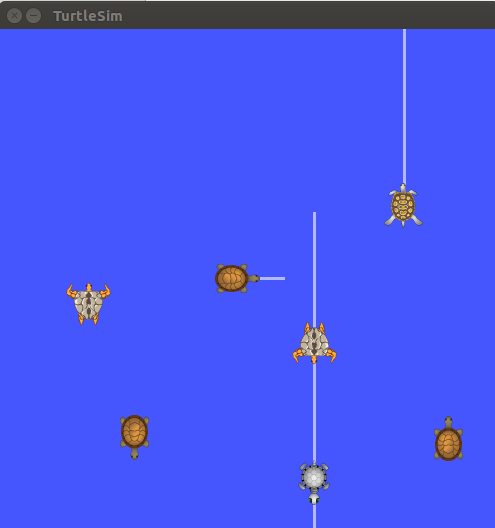
Run 4: T3 runs close to main and stop then run away



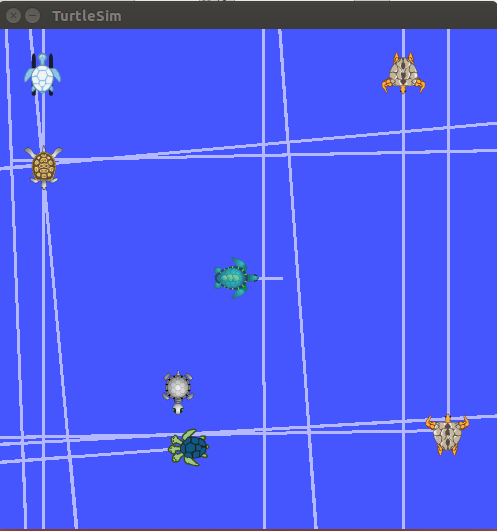
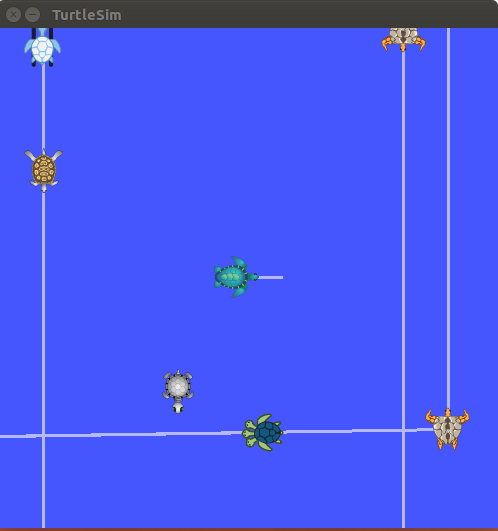
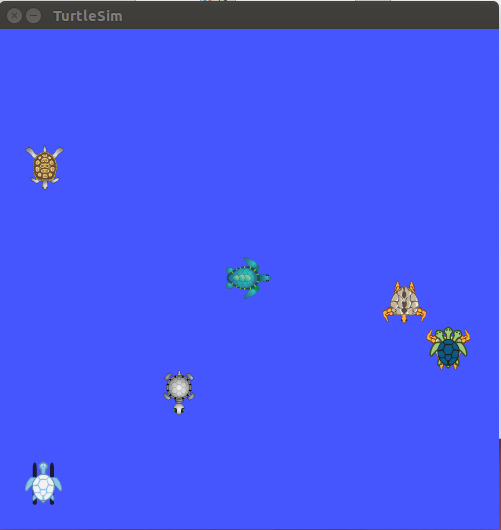
Run 4: T3 runs crazy



Run 5: T3 runs in square loop



Run6:



movePattern3

**void TurtlesBehavior::movePattern3(HW::Turtle& tturtle, ros::Publisher& tpub, int speed, int distance, int rdir)** {

double desired\_angle\_radians;

if (rdir == 0)

desired\_angle\_radians = PI;

else if (rdir == 1)

desired\_angle\_radians = -PI;

else

desired\_angle\_radians = 2\*PI;

if (HW::turtleExist(tturtle.turtlename)) { //is this turtle alive?

cout << "P3: " << tturtle.turtlename << " distance=" << distance;

cout << " speed=" << speed;

cout << " rotationAngle= " << desired\_angle\_radians;

HW::moveDistance(speed, distance, true, tturtle, tpub);

setDesiredOrientation(desired\_angle\_radians, tturtle, tpub);

HW::moveDistance(speed, distance, true, tturtle, tpub);

setDesiredOrientation(desired\_angle\_radians, tturtle, tpub);

};

};

**void setDesiredOrientation (const double desired\_angle\_radians, const HW::Turtle& t, const ros::Publisher& velpublisher)**{

double relative\_angle\_radians = desired\_angle\_radians - t.pose.theta;

bool clockwise = ((relative\_angle\_radians<0)?true:false);

//cout<<desired\_angle\_radians <<","<<t.pose.theta<<","<<relative\_angle\_radians<<","<<clockwise<<endl;

rotate (degrees2radians(ANGULAR\_SPEED), abs(relative\_angle\_radians), clockwise, velpublisher);

}

**void moveDistance(const double speed, const double distance, const bool isForward, HW::Turtle& turtle, const ros::Publisher& velpublisher)** {

geometry\_msgs::Twist velmsg;

if (isForward)

velmsg.linear.x = abs(speed);

else

velmsg.linear.x = -abs(speed);

velmsg.linear.y = 0;

velmsg.linear.z = 0;

velmsg.angular.x = 0;

velmsg.angular.y = 0;

velmsg.angular.z = 0;

double t0 = ros::Time::now().toSec();

double t1;

double current\_distance = 0.0;

ros::Rate loop\_rate(LOOP\_SPEED);

do{

velpublisher.publish(velmsg);

t1 = ros::Time::now().toSec();

**current\_distance = speed \* (t1-t0); //distance = delta\_time\*speed**

ros::spinOnce();

loop\_rate.sleep();

cout << turtle.turtlename << ":" << turtle.pose.x << "," << turtle.pose.y << "," << turtle.pose.theta << endl;

if (isOffBoundary(turtle.pose.x, turtle.pose.y)) {

//cout << " .revx.: " << velmsg.linear.x << endl;

//cout << "pose: (" << turtle.pose.x << "," << turtle.pose.y << ")" << endl;

velmsg.linear.x = -1\*velmsg.linear.x;

velpublisher.publish(velmsg);

sleep(1);

};

} while (current\_distance < distance);

velmsg.linear.x =0;

velpublisher.publish(velmsg);

cout << " deltaTime=" << (t1-t0) << endl;

}