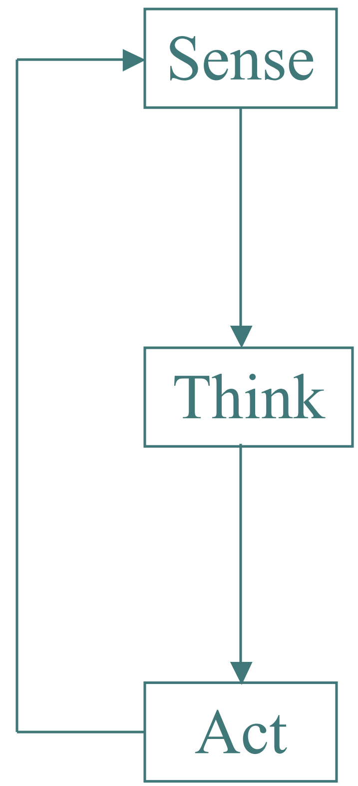
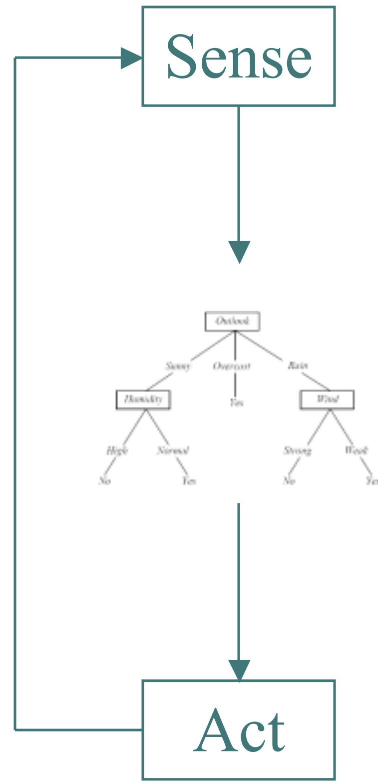
Sense, Think, Act

- Sense
  - Gather input sensor changes
  - Update state with new values
- Think
  - *Decide what to do*
- Act
  - Execute (any changes to) actions



Questions

- Can we learn appropriate decision making by observation?
- If so, can the design of the ‘think step’ or decision module be automated?

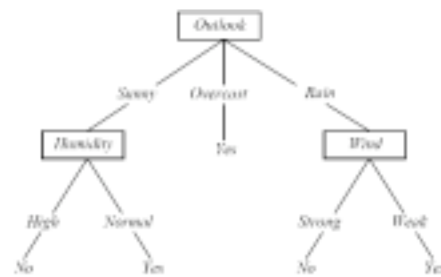


⇒ Decision trees are ideal for this, easy to learn and transparent.

Building Decision Trees

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Train



Training Dataset

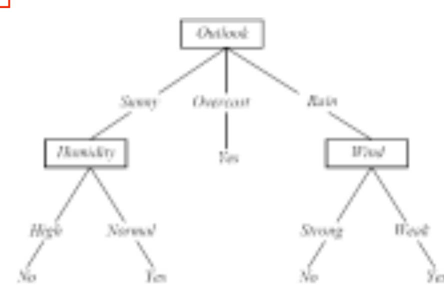
DT

How good is the the learned DT?

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes

Testing Dataset

Test



DT

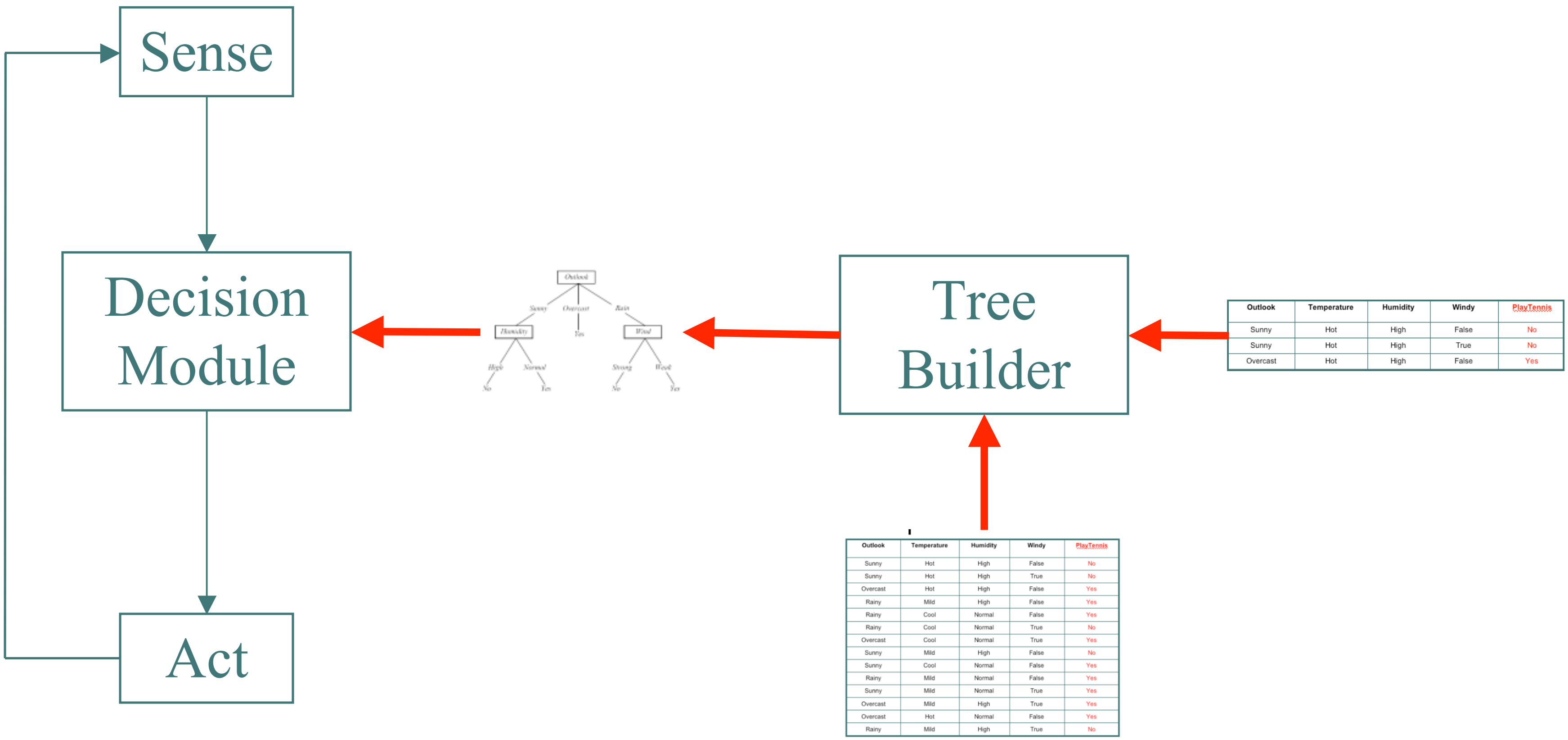
Train & Test Datasets

- The training dataset encodes the observations of the task we want the decision tree to learn.
- The test dataset looks just like the training dataset except that the decision tree algorithm never sees this data during DT construction time.
- We apply the DT to the *independent* attributes of the test set …
- …and then compare the answer of the decision tree with the value of the target attribute for each row in the test set, respectively

Where do these datasets come from?

- In games they are usually hand constructed by the AI engineers
- In other areas they tend to be observations/records collected over many years

Integrating DTs into QII



Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

The ID3 Tree Builder

Independent Attributes

Target Attribute

Observations
or
Training Examples

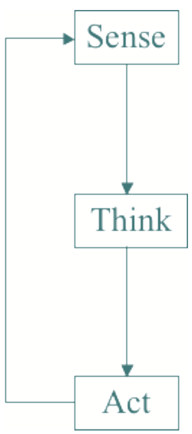
OUTLOOK	TEMP	HUMIDITY	WINDY	PLAY
sunny	hot	high	FALSE	no
sunny	hot	high	TRUE	no
overcast	hot	high	FALSE	yes
rainy	mild	high	FALSE	yes
rainy	cool	normal	FALSE	yes
rainy	cool	normal	TRUE	no
overcast	cool	normal	TRUE	yes
sunny	mild	high	FALSE	no
sunny	cool	normal	FALSE	yes
rainy	mild	normal	FALSE	yes
sunny	mild	normal	TRUE	yes
overcast	mild	high	TRUE	yes
overcast	hot	normal	FALSE	yes
rainy	mild	high	TRUE	no

NOTE: the ID3 tree builder always assumes that the last attribute
Is the target attribute.

The Decision Module

```
class DecisionModule
{
        // attribute values

        // values for OUTLOOK
        public static final int sunny = 0;
        public static final int overcast = 1;
        public static final int rainy = 2;
        // values for TEMP
        public static final int hot = 0;
        public static final int mild = 1;
        public static final int cool = 2;
        // values for HUMIDITY
        public static final int high = 0;
        public static final int normal = 1;
        // values for WINDY
        public static final int FALSE = 0;
        public static final int TRUE = 1;
        // values for PLAY
        public static final int no = 0;
        public static final int yes = 1;

        // decision function

        public static int tree(int OUTLOOK,
                               int TEMP,
                               int HUMIDITY,
                               int WINDY) {

                if( OUTLOOK == sunny ) {
                        if( HUMIDITY == high ) {
                                return no;
                        }
                        if( HUMIDITY == normal ) {
                                return yes;
                        }
                }
                if( OUTLOOK == overcast ) {
                        return yes;
                }
                if( OUTLOOK == rainy ) {
                        if( WINDY == FALSE ) {
                                return yes;
                        }
                        if( WINDY == TRUE ) {
                                return no;
                        }
                }
                return -1;
        }
}
```

Teaching Quagents to Walk



- Assume we have an infrastructure that can sense the environment appropriately.
- Also assume that we have an infrastructure that can adequately interpret commands.
- Goal: Learn the ‘think step’ - learn the decision module.

Building the Decision Module

NOTE: you need to build a table in following format:

```
Left            Right           Front           Navigate
/////////////////////////////////////////////////////////
lclear          rclear          fclear          walk
lclear          rclear          fblocked        left
lblocked        rclear          fblocked        right
```

Attribute Values:

Left: lclear, lblocked
Right: rclear, rblocked
Front: fclear, fblocked
Navigate: walk, left, right

To build the decision module run it through the ID3 tree builder:

java -classpath . ID3 traindata.txt

Building the Decision Module

```
class DecisionModule
{
        // attribute values

        // values for Left
        public static final int lclear = 0;
        public static final int lblocked = 1;
        // values for Right
        public static final int rclear = 0;
        public static final int rblocked = 1;
        // values for Front
        public static final int fclear = 0;
        public static final int fblocked = 1;
        // values for Navigate
        public static final int walk = 0;
        public static final int left = 1;
        public static final int right = 2;

        // decision function

        public static int tree(int Left,int Right,int Front) {
                if( Front == fclear ) {
                        return walk;
                }
                if( Front == fblocked ) {
                        if( Left == lclear ) {
                                return left;
                        }
                        if( Left == lblocked ) {
                                return right;
                        }
                }
                return -1;
        }
}
```

The ‘Learner’ Infrastructure

```
// loop forever -- that is until the bot dies of old age
try {
    while(true) {
        q.rays(4);
        cmd = think(q.events());
        navigate(cmd);
    }
} catch (QDiedException e) { // the quagent died -- catch that exception
    System.out.println("bot died!");
}
```

The ‘Learner’ Infrastructure

```
public int think(Events events)
        throws Exception
{
    String[] words = parseRaysEvents(events);

    // this is what the event looks like:
    // OK  (ask rays 4)
    //     1 worldspawn 379.969 0 0
    //     2 worldspawn -7.62939e-006 247.969 0
    //     3 player -43.9688 0 0
    //     4 worldspawn 0 -247.969 0

    double xf = Double.parseDouble(words[6]);
    double yf = Double.parseDouble(words[7]);
    double df = Math.sqrt(xf*xf + yf*yf);
    int f = df > blockDistance ? DecisionModule.fclear : DecisionModule.fblocked;

    double xl = Double.parseDouble(words[11]);
    double yl = Double.parseDouble(words[12]);
    double dl = Math.sqrt(xl*xl + yl*yl);
    int l = dl > blockDistance ? DecisionModule.lclear : DecisionModule.lblocked;

    double xr = Double.parseDouble(words[16]);
    double yr = Double.parseDouble(words[17]);
    double dr = Math.sqrt(xr*xr + yr*yr);
    int r = dr > blockDistance ? DecisionModule.rclear : DecisionModule.rblocked;

    //// CALL THE DECISION MODULE ////
    int n = DecisionModule.tree(l,r,f);

    return n;
}
```

The 'Learner' Infrastructure

```
public void navigate(int cmd)
        throws Exception
{
    // interpret the commands from the decision module
    switch(cmd) {
    case DecisionModule.left:
        q.turn(90);
        break;
    case DecisionModule.right:
        q.turn(-90);
        break;
    case DecisionModule.walk:
        q.walk(100);
        break;
    default:
        throw new Exception("unknown navigation command");
    }
}
```