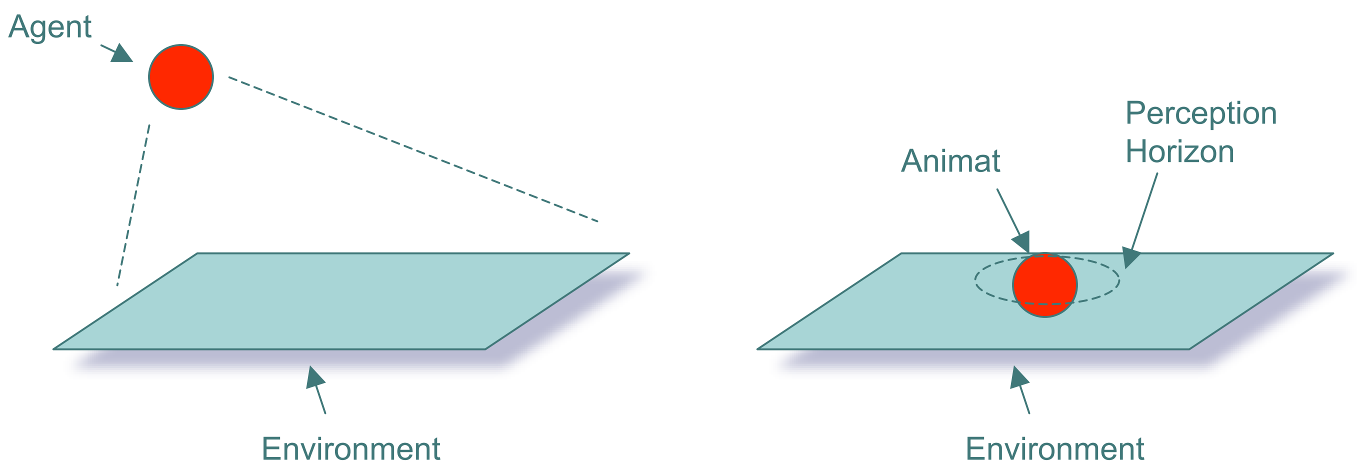
Embodiment

- A traditional agent perceives <u>global state information</u>.
- An animat can only perceive information embedded in its <u>immediate local vicinity</u> through its "senses."



Embodiment

- The exact nature of the senses depends on the virtual world the animat inhabits.
- Game engines tend to emulate aspects of the physical world, therefore animats in games tend to have senses that are modeled after our own or other creatures in our physical world:
  - Vision
  - Sense of touch (bumping into things, etc.)
  - Sense of locomotion (e.g., direction, speed)
  - Introspection, well-being

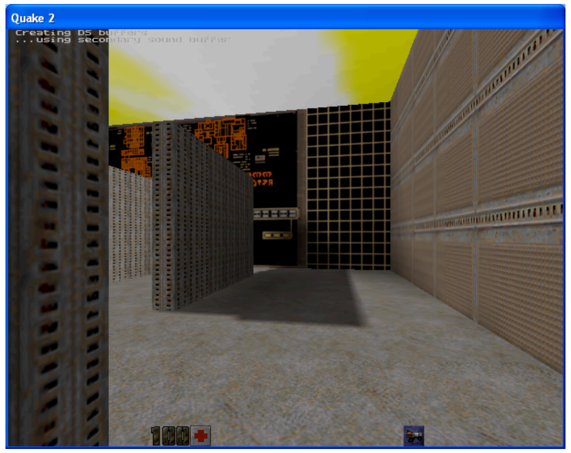
☞ As a consequence of embodiment the animat has only limited information to base decisions on.

Embodiment

- Embodiment entails having a *body* (albeit simulated for animats in virtual worlds):
  - We need to actively move the body expending energy and plan our route.
  - The body can only move according to the abilities of the animat.
  - The body ages.
  - The body occupies space – no two things can exist in the same location in the simulated world.

The Quagent World

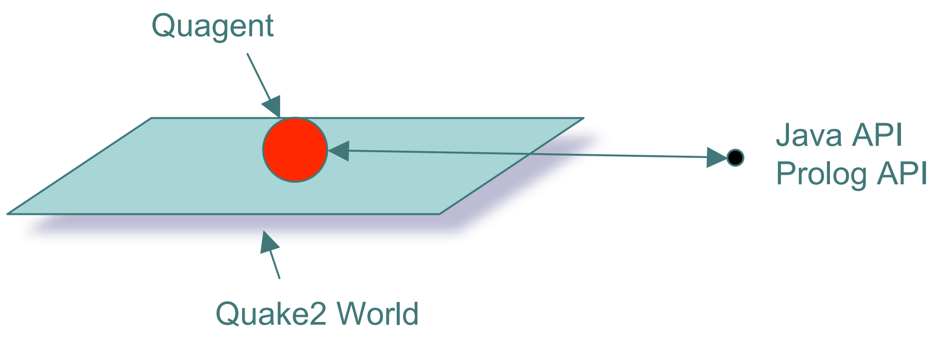
- Worlds are simulated using the Quake2 game engine.
    - Two simple worlds are provided to you by default:
        - Empty Room
        - Obstacle Room
    - You will get the chance to build your own world (level design) later.
    - Or explore other more complicated worlds using your animats.



```
Creating DS buffers
...using secondary sound buffer
```

Quagents

- Quagent ≡ Quake Agent
- A better term would have been Quanimat (but that sounds very strange)
- A quagent is an animat that inhabits a simulated world.
- The unique property of quagents is that we have access to their "brains."



Quagents

- The Java and Prolog APIs provide access to the animat body:
  - Senses
  - State
  - Locomotion
- We can write code to control the animats

Prolog ≡ **Pro**gramming in **Log**ic API ≡ Application Programming Interface

Quagent Java API

```
class Quagent {
    // spawn a new quagent
    public Quagent () throws Exception {...}
    public Quagent (String hostName) throws Exception {...}

    // Actions
    public void walk(int distance) throws Exception {...}
    public void turn(int angle) throws Exception {...}
    public void pickup(String itemName) throws Exception {...}
    public Events drop(String itemName) throws Exception {...}

    // Perception
    public void radius(float radius) throws Exception {...}
    public void rays(int no_of_rays) throws Exception {...}
    public void cameraon() throws Exception {...}
    public void cameraoff() throws Exception {...}

    //Proprioception
    public void where() throws Exception {...}
    public void inventory() throws Exception {...}
    public void wellbeing() throws Exception {...}

    //Event Retrieval
    public Events events() throws Exception {...}

    // Abandon
     public void close() throws Exception {...}
}
```

Quagent Java API

- The Java API is *asynchronous*.
- The reason: quagents can generate *multiple, asynchronous messages* called *events* to the brain as a result of a command or the environment.
- These events are captured in the Events object.
- Possible events:
  - OK ([Echo])
    - echos for positive confirmation, as in a submarine movie
  - ERR ([Echo]) [Error Description]
    - can't begin to do command
  - TELL [Event] [Parameters]
    - asynchronous message from quagent: provides information about it’s internal state and other parameters

Quagent Java API

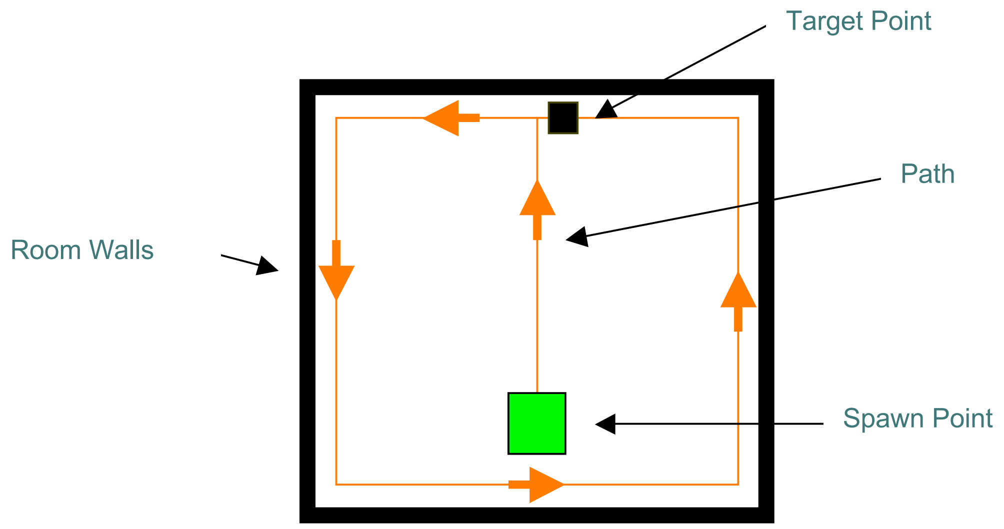
```
// connect to a new quagent
Quagent q = new Quagent();

try {
    int dist = (int)(Math.random()*100.0);

    // walk command
    q.walk(dist);

    // get position
    q.where();

    // get wellbeing
    q.wellbeing();

    // get events
    printEvents(q.events());

} catch (QDiedException e) {
    System.out.println(“quagent died!");
}

// abandon quagent
q.close();
```

Assignments

- Read Chapters: 1,2 (Alex)
- Programming Assignment, due Wednesday 2/3 at 10pm, see assignment sheet.

Programming Assignment #1 – The Warm-Up Lap

Write a quagent program in Java that navigates the ‘Empty Room’ world in the following fashion.



Java API Details

- Walk command, ‘walk(distance)’
  - Desire quagent to start walking in current direction. Bot will, at best, start walking in the right direction and silently stop when distance is reached.
  - **GOTCHA:** The quagent is really a bounding box and some graphics that change, often in a cycle, when it is doing something, for example walking. The "realistic" look requires that the quagent move unequal forward distances between the various images that depict its action. Thus your quagents take "different sized steps" during their walk cycle. Therefore, they only approximate the distance in the walk command, and if you are counting steps (ticks) or something to compute your own distances, you can be surprised.
  - Response: OK (do walkby <distance>)
  - Error: ERR <error message>
  - Generates a TELL event to inform you how far the quagent actually walked.
    - TELL STOPPED <actual distance>

Java API Details

- Turn command, ‘turn(angle)’
  - Angle is in degrees, + (left turn) or - (right turn). Changes current yaw (orientation).
  - Response: OK (do turnby <angle>)
  - Error: ERR <error message>
- Pickup command, ‘pickup(“tofu”)’
  - Immediately picks up one of the named items if quagent is within reach of the item.
  - Response: OK (do pickup <item>)
  - Error occurs if not close enough: ERR <error message>
- Drop command, ‘drop(“tofu”)’
  - Immediately puts down one of the named items in the quagent’s inventory.
  - Response: OK (do drop <item>)
  - Error occurs if not in inventory: ERR <error message>

Java API Details

- Radius command, ‘radius(100.0)’
  - What items are within the given radius? (There is a system-imposed max-radius).
  - Response: OK (ask radius <radius>) [number of objects] ([object name] [relative position])*
    - here [relative position] is an (x,y,z) three-vector of relative distances.
    - Example: OK (ask radius 100.0) 2 GOLD -20.0 30.0 0 TOFU -320 -100 0
  - Error: ERR <error message>
- Rays command, ‘rays(4)’
  - What entities are surrounding quagent in some set of directions. If number is one, ray's direction is in quagent’s current direction. The command shoots out rays evenly distributed on a circle around the robot.
  - Response: OK (ask rays <#>) ([ray_number] [object_name] [relative_position])+
    - relative_position is as in the radius command. The "world_spawn" entity is a wall or other game structure.
    - Example: OK (ask rays 2) 1 world_spawn 315.0 277.1 0.0 2 TOFU 200 100 0
  - Error: ERR <error message>

Java API Details

- Cameraon command, ‘cameraon()’
  - Normally the terminal shows the view from the (first-person) client. This puts the camera on the quagent.
  - Response: OK (do cameraon)
  - Error: ERR <error message>
- Cameraoff command, ‘cameraoff()’
  - Puts camera on client (first-person).
  - Response: OK (do cameraon)
  - Error: ERR <error message>

Java API Details

- Where command, ‘where()’
  - Where is the quagent, how is it oriented, moving how fast?
  - Response: OK (do getwhere) [world state]
    - where [world state] is a vector of coordinates and a velocity: (world_x, world_y, world_z, roll, pitch, yaw, velocity).
    - Example: OK (do getwhere) 124.5 -366 492 0 0 0 1
  - Error: ERR <error message>
- Inventory command, ‘inventory()’
  - What is quagent holding?
  - Response: OK (do getinventory) [Inventory]
    - where [Inventory] is (inventory item name)*
    - Example: OK (do getinventory) tofu data
  - Error: ERR <error message>

Java API Details

- Wellbeing command, ‘wellbeing()’
  - How is quagent doing in its life?
  - Response: OK (do getwellbeing) [well being].
    - Where [well being] is a vector of strings giving the numerical values of (age, health, wealth, wisdom, energy).
    - Example: OK (do getwellbeing) 720 0 0 0 925.6
  - Error: ERR <error message>
  - Generates a ‘TELL DYING’ event when the quagent dies either of old age or of low energy.
  - The ‘TELL DYING’ event generates a ‘QDiedException’ Java exception that you can catch and process.
- Close command, ‘close()’
  - Abandons the quagent in the quagent world; the quagent dies immediately (without generating an exception)

Programming Tricks

Running the Simple quagent program:

1) start quake2/empty room – by clicking on the appropriate Q2 icon
2) open a command shell window
3) cd into the directory where the java api and example code is.
4) compile all the java code: javac -classpath . *.java
5) run the simple example: java -classpath . Simple

OR

Copy all the (appropriate) source files into a project in your favorite IDE (Eclipse, MetroWorks, etc) and compile and run your program from there.

☞ Make sure you use the code in folder “Java V3”.

Programming Tricks - Template

Quagent Application Template

```
// spawn a new quagent in the world
q = new Quagent();

// start a try/catch block and execute the quagent communication
try {
        // do quagent stuff
    }
} catch (QDiedException qe) {
    // the quagent died
    System.out.println("bot died!");
    // ...
} catch (Exception e) {
    // something else bad happened, execute recovery code
    System.out.println(e.getMessage());
    // ...
}

// we are all done with this quagent -- abandon
q.close();
```

Programming Tricks - Events

- Events are integral to asynchronous programming
- Quagents have a very simple event structure:

```
class Events {
          // constructor
    Events() {...}
          // add an event string to the event object
    public void add(String s) {...}
          // return the number of events in the object
    public int size () {...}
          // return the event at index ix
    public String eventAt(int ix) {...}
}
```

Programming Tricks - Events

- Events are arrays of strings returned from the game engine,

```
Events e = q.events();
int n = e.size();
String fst = e.eventAt(0);
String lst = e.eventAt(n-1);
```

- A typical event object might look something like this

```
0: "OK (do walkby <distance>)"
1: "TELL STOPPED <actual distance>"
2: "OK ( ask rays 2 ) 1 world_spawn 315.0 277.1 0.0 2 TOFU
200 100 0"
```

Programming Tricks - Events

Parsing Event Codes

```
public void parseEventCodes(Events events) {

            // get the individual events from the event object
            for (int ix  = 0; ix < events.size(); ix++) {

                // get the event codes
                String eventString = events.eventAt(ix);
                String[] tokens = eventString.split("\\s+");
                String eventCode = tokens[0];
                String eventVal = tokens[1];
                String eventParam = tokens[2];

                // do something useful with the codes
                if (eventCode.equals("TELL")) {
                      if (eventVal.equals("STOPPED")) {
                          System.out.println("Moved: " + eventParam);
                      }
                } else if (eventCode.equals("OK")) {
                      System.out.println("return status OK");
                } else if (eventCode.equals("ERR")) {
                      System.out.println("return status --> error");
                } else {
                      continue;
                }
            }
      }
}
```

Programming Tricks - Events

- Finding events that share a keyword

```
public Events findEvents(Events events, String kword) {

        Events newEvents = new Events();

        for (int ix  = 0; ix < events.size(); ix++) {
            String e = events.eventAt(ix);
            if (e.indexOf(kword) >= 0) {
                newEvents.add(e);
            }
        }

        return newEvents;
    }
```

Programming Tricks - Events

- Computing the distance an animate walked

```
public double parseWalkEvent(String eventString) {
    // TELL STOPPED <number>
    String[] tokens = eventString.split(“\\s+”);

    return Double.parseDouble(tokens[2]);
}
```

Programming Tricks - Events

- Computing distances with Rays

```
public Double rayDistance(String eventString)
{
        // NOTE: only works for single ray commands
        // this is what the event looks like:
        // OK  (ask rays 1) 1 worldspawn 379.969 54.342 0

        // NOTE: parens are considered white space.

        String[] tokens = eventString.split("[()\\s]+");

        double x = Double.parseDouble(words[6]);
        double y = Double.parseDouble(words[7]);

        double distance = Math.sqrt(x*x + y*y);

        return distance;
}
```

Programming Tricks

Configuring the Quagent World: quagent.config

```
quagent { (TYPE [Botname])? ([variable] [value])* }

[variable] is one of
        LIFETIME,
        INITIALWISDOM,
        INITIALENERGY,
        INITIALWEALTH,
        INITIALHEALTH,
        ENERGYLOWTHRESHOLD,
        AGEHIGHTHRESHOLD,
for which [value] is a string to be interpreted as a number, or
it can be this keyword followed by the coordinates:
        INITIALLOCATION [x] [y] [z]

Available Quagents:
        soldier
        berserk
        gunner
        infantry
        parasite

Examples:
        quagent {type soldier initiallocation 128 -236 24.03}
        quagent {type parasite lifetime 5000 initialenergy 1000}
        quagent {type gunner lifetime 6000 initialenergy 2000}
```

NOTE:
For each quagent spawned you will need a separate ‘quagent’ statement in the config file otherwise the quagent will default to mitsu/soldier

Programming Tricks

Configuring the Quagent World: quagent.config

```
[object] [x] [y] [z]
where [object] is one of the following
        tofu,
        data,
        battery,
        gold,
        kryptonite`
for which [x] [y] [z] are strings to be interpreted as numbers.

Examples:
        data -155 256 0
        gold 0 0 0
```