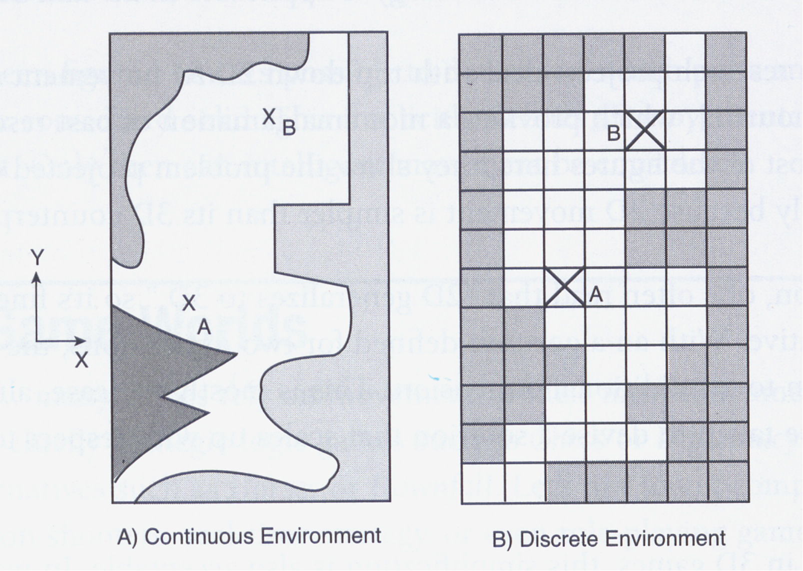
Navigation

- Navigation is the process of purposefully steering the course of an entity through a space.
- Navigation differs from plain movement
  - Plain movement could be due to such occurrences like an object falling off a cliff.

Navigation

- A game world describes a space.

Discrete
vs.
Continuous



A) Continuous Environment

B) Discrete Environment

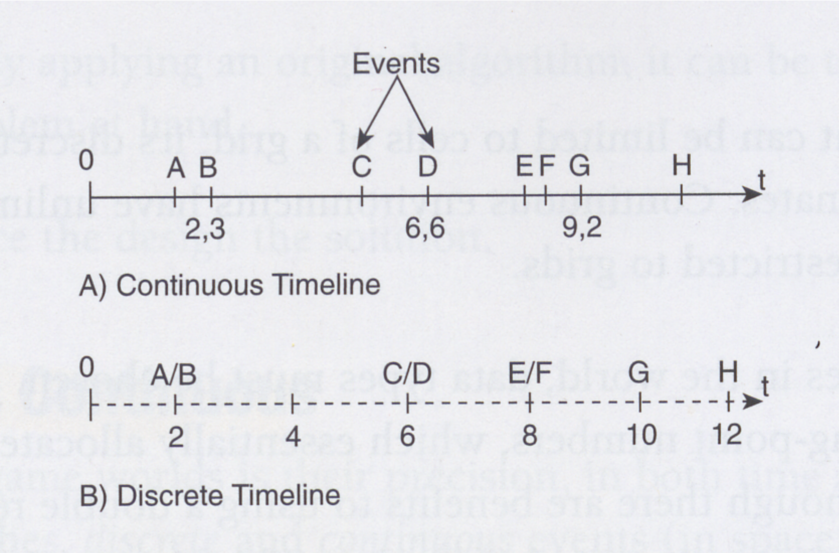
☞This has consequences on where items can be placed and how agents can move from one position to the next.

☞ Quake uses continuous space

Navigation

- A game world describes time.

Discrete
vs.
Continuous



A) Continuous Timeline

B) Discrete Timeline

☞This has consequences on how actions are perceived – smooth vs. choppy.

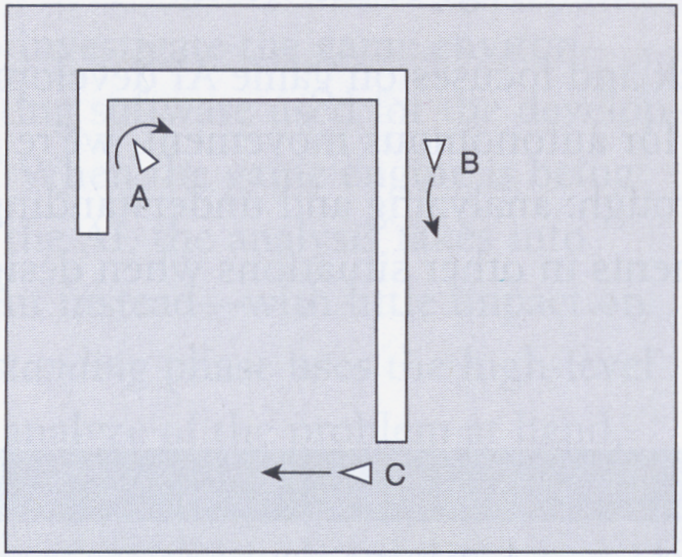
☞ At the human perception level Quake uses continuous time - intervals are a couple of milliseconds.

Navigation

- We want navigation to be
  - Realistic
    - avoid doing silly things
  - Efficient
    - it cannot be computationally expensive
  - Reliable
    - the same navigation strategies should work in many different scenarios
  - Purposeful
    - it should serve some perceived goal

Navigation

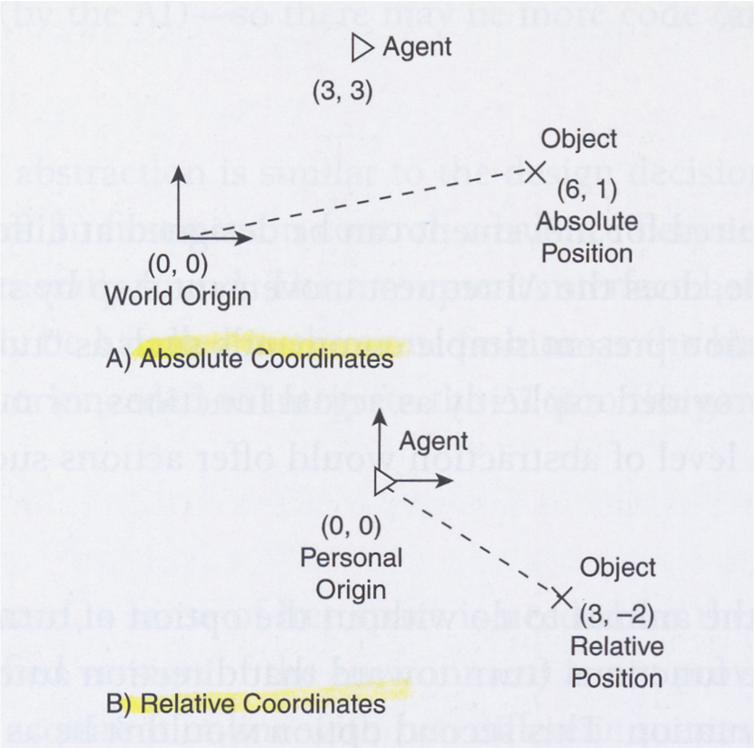
- Example Scenarios – “Obstacle Avoidance Maneuvers”



A) Complete Turn in Corner
B) Partial Turn Near Wall
C) No Obstacles, Forward Movement

Navigation - Options

- Agent Context

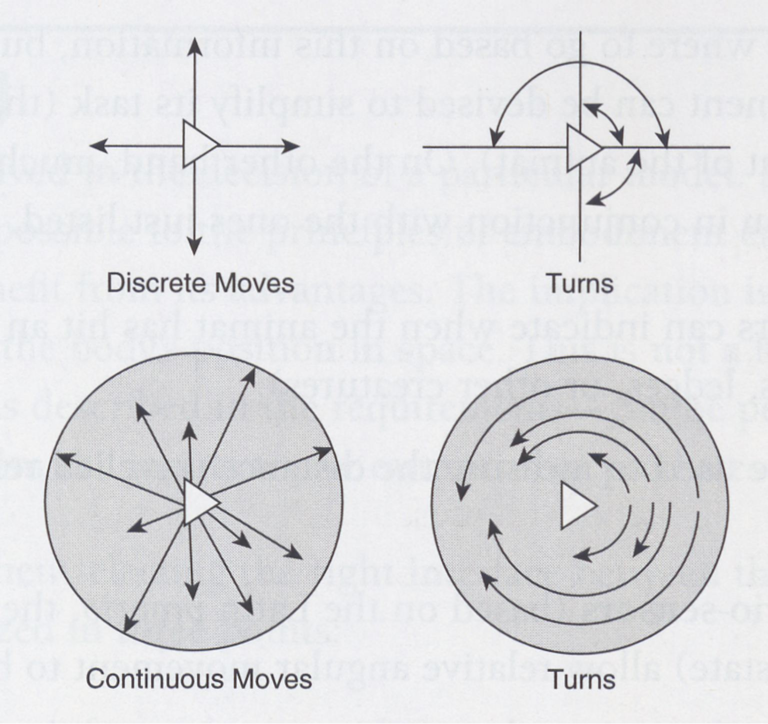


☞ In the quagent API radius and rays calls return results in relative coordinates

☞ The where function returns results in absolute coordinates

Navigation - Options

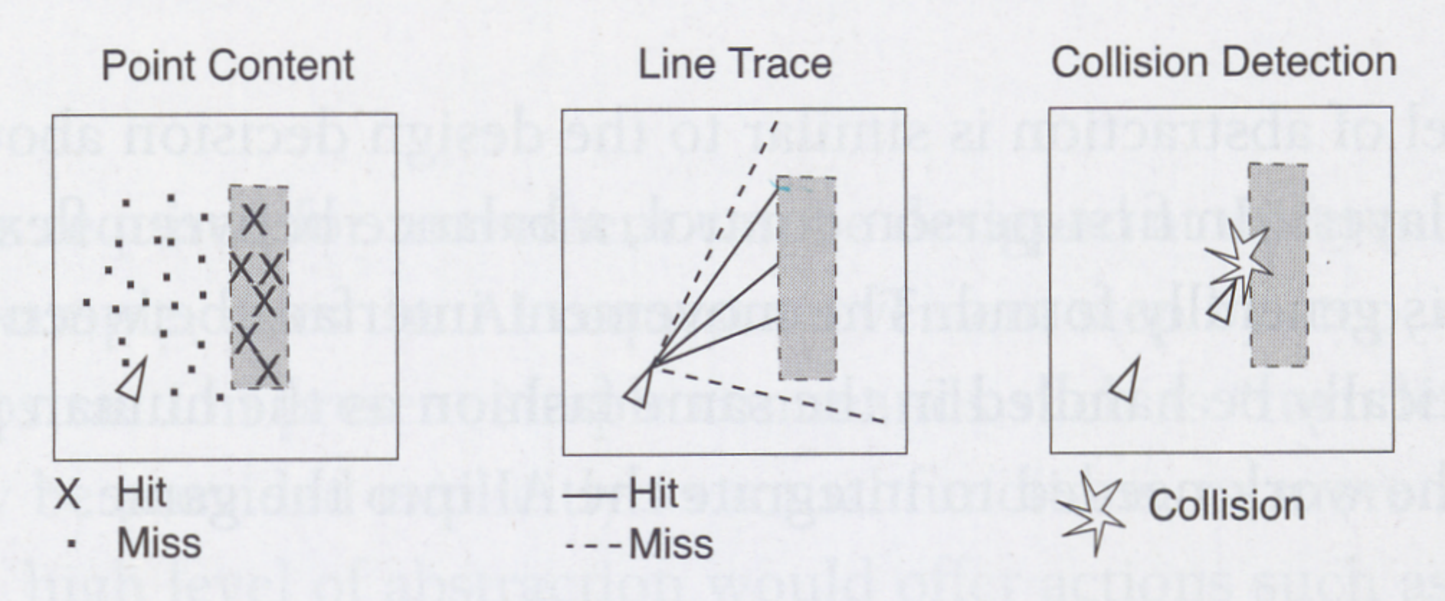
- Discrete vs. Continuous Actions



☞ In the quagent world all actions are continuous

Navigation – Options

- Senses



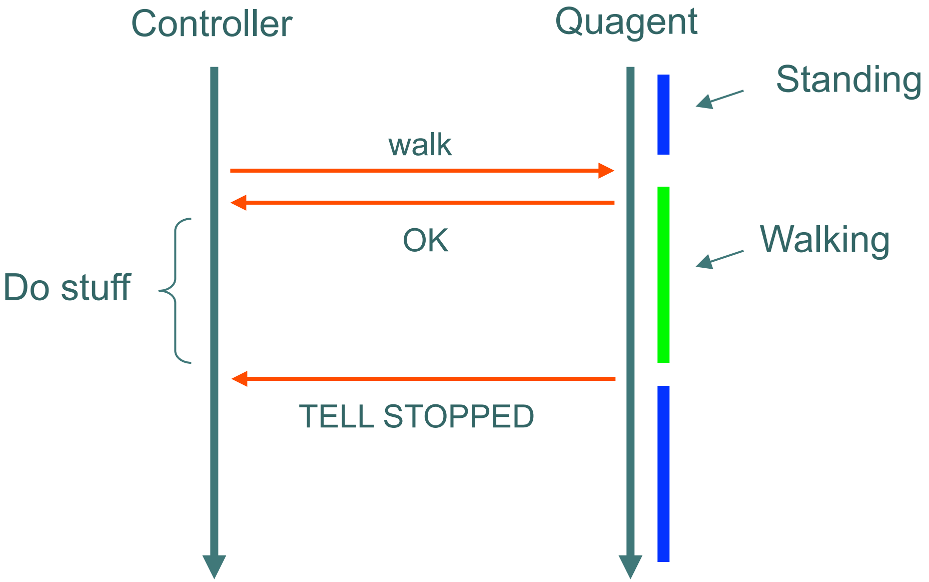
☞ Quagents implement point content with the radius command

☞ Quagents implement line trace with the rays command

☞Quagents implement collision detection with the TELL STOPPED event

Interprocess Communication

Example: `... q.walk(256); ...`



```
...
Events events = null;
bool stopped = true;
...
q.walk(256);
stopped = getStopped(q.events());

while(!stopped) {
   // do stuff
   events = q.events();
   stopped = getStopped(events);
}
...
```

Event Polling

NOTE:
getStopped will return true if it finds the ‘TELL STOPPED’ event, otherwise it will return false.

Steering a Quagent

- Idea:
  - Tell the quagent to walk a very large distance
  - Then use ‘rays’ to see if there are obstacles
  - keep exchanging messages with the quagent about navigating possible obstacles

IPC

```
class Asynch extends Quagent {

    static final int DIST = 20;

    public static void main(String[] args) throws Exception {
                new Asynch();
    }

    Asynch () throws Exception {
                super(); // run the constructer of the super class

                try {
                            this.walk(5000);
                            while(true) {
                                        // sense
                                        this.rays(1);
                                        Events e = this.events();
                                        // think & act with event handlers
                                        handleRays(e);
                                        handleStopped(e);
                                        // give the engine a chance to do something
                                        Thread.currentThread().sleep(100);
                            }
                } catch (QDiedException e) { // the quagent died -- catch that exception
                            System.out.println("bot died!");
                }

                this.close();
    }
```

IPC

```
public void handleRays(Events events) throws Exception {
        for (int ix = 0; ix < events.size(); ix++)
        {
                String e = events.eventAt(ix);

                if (e.indexOf("rays") >= 0)
                {
                        // 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 not included in tokens
                        String[] tokens = e.split("[()\\s]+");

                        double x = Double.parseDouble(tokens[6]);
                        double y = Double.parseDouble(tokens[7]);
                        double distance = Math.sqrt(x*x + y*y);

                        System.out.println("Distance: " + distance);

                        // if the distance is less than DIST ticks then turn 90 degrees left
                        if (distance < DIST)
                                this.turn(90);
                }
        }
  }
}
```

IPC

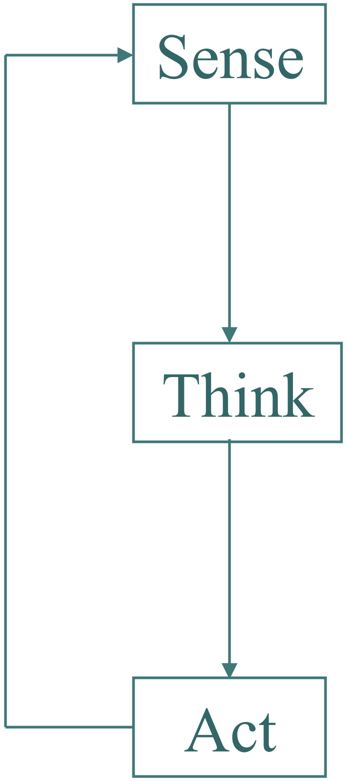
```
public void handleStoppedEvents events) throws Exception {
        for (int ix = 0; ix < events.size(); ix++)
        {
                String e = events.eventAt(ix);

                if (e.indexOf(”STOPPED") >= 0)
                {
                        // probably bumped into something
                        this.turn(180);
                        // start walking again
                        this.walk(5000);
                }
        }
  }
}
```

The “Sense, Think, Act” Loop

- The previous example highlighted the fact that in many cases quagent control can be embedded in a loop
- The loop will iterate over three kinds of activities:
  - Sensing
  - Thinking (computing)
  - Acting

The “Sense, Think, Act” Loop

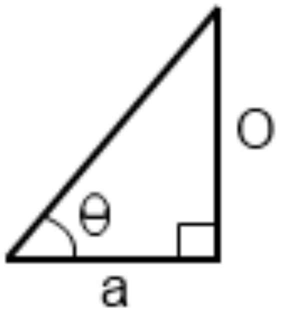
- Sense
  - Gather input sensor changes
  - Update state with new values
- Think
  - *Decide what to do*
- Act
  - Execute (any changes to) actions



Programming Tricks

- Navigation
  - The radius and rays command return relative positions
  - That means, once you have found an object you need to calculate angle and distance to reach it
  - With rays this is trivial because rays only “appear in certain angles”
  - With radius command it is a little bit more difficult…consider…

Programming Tricks

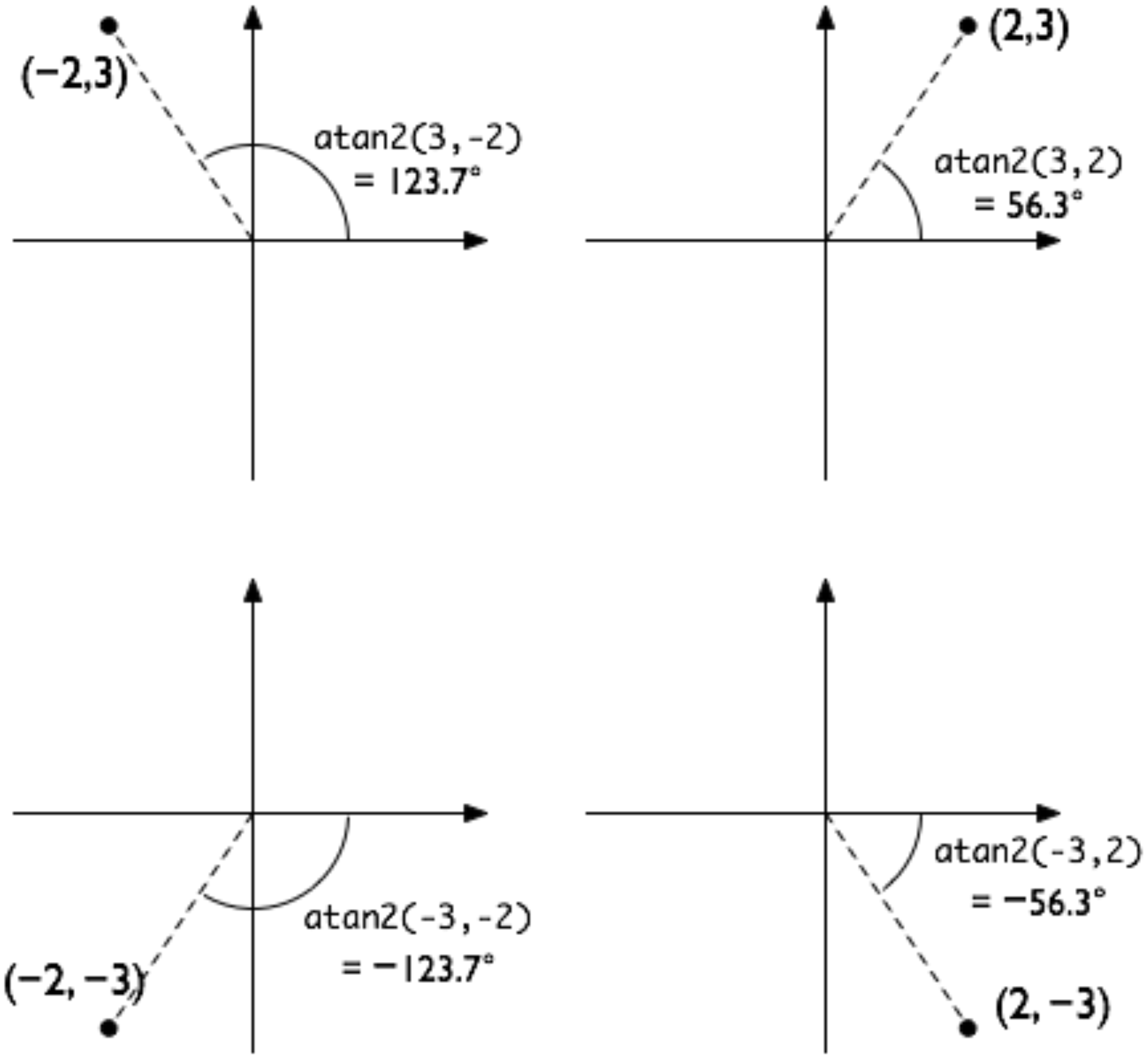


- The arc-tan can compute the angle given the sides of a triangle:

  atan(o/a) = θ

source: http://gamedev.stackexchange.com/questions/14602/what-are-atan-and-atan2-used-for-in-games

Programming Tricks



Programming Tricks

- To compute an angle in you need to turn you will need to experiment with the ‘where’ command giving you the absolute position of the quagent and the **yaw** – the angle of rotation around the vertical axis (z-coordinate) of the quagent.

Programing Exercise #2

- Randomly place an object into the Empty Room using the config file and have a quagent find it and pick it up – no hard coding allowed, you will need to search, find, and navigate to it in order to pick the item up.
- Next, put multiple objects into the Empty Room and repeat the above for multiple objects in a row – ie. continue searching and pickup until the quagent dies of old age.
- Next, do the same things in the Obstacle Room.