# Presentatie Michiel

michielborkent@gmail.com

Informaticastage najaar 2004

# Inhoud presentatie:

- Wie ben ik?
- Over mijn stage
- Informatie over Lisp

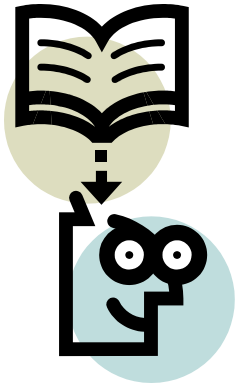
# Wie/wat ben ik?

- Student Technische Informatica, Universiteit Twente

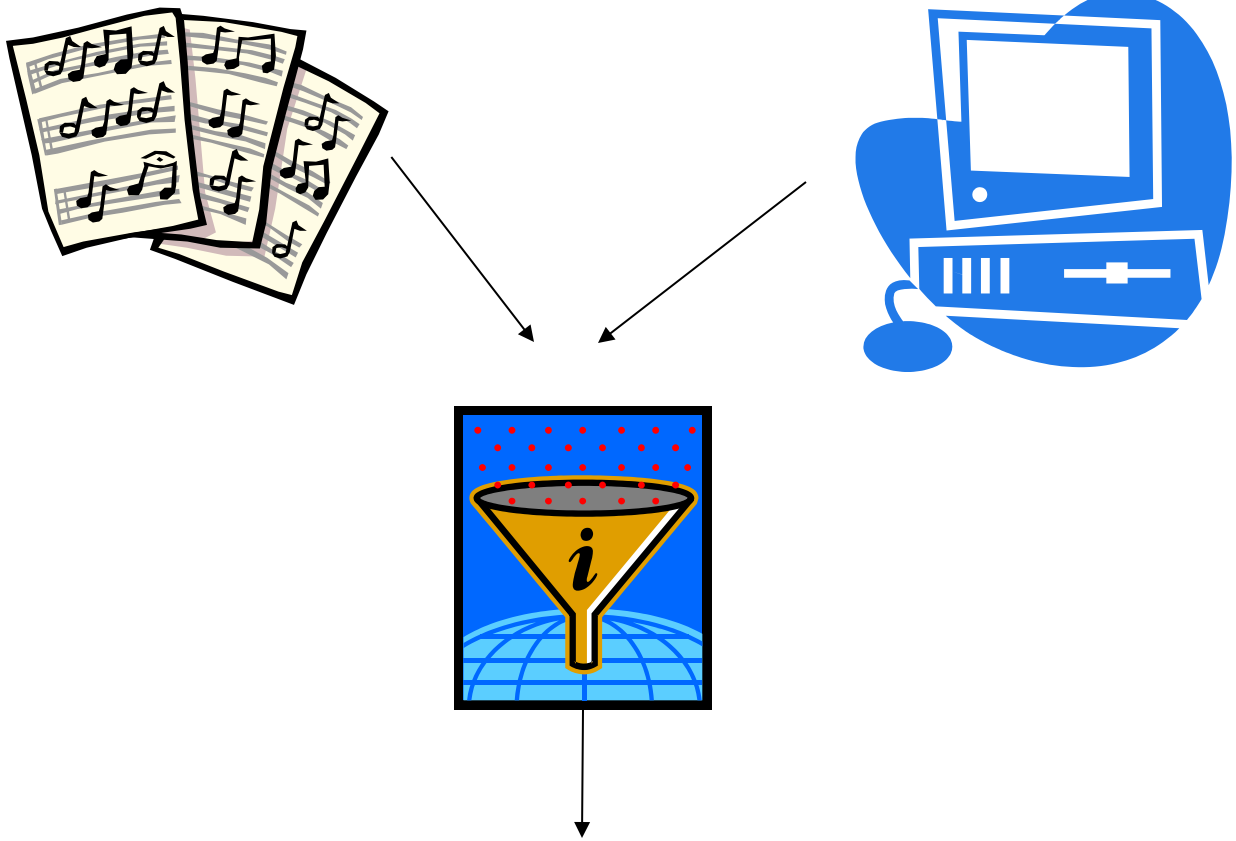

Michiel, Hongarije 1991

# O.a. grote interesse in muziek

Op zoek naar een stage. Idee?



# Combineren van interesses!




stage

# Op zoek naar stage

- Zoeken in Google

op de woorden: *muziek informatica stage*.

- Uitkomst MMM: **Music Mind Machine**.
- **Machine** = informatica, **Music** = andere interesse. Puzzel (bijna) compleet!

…

# Michiel doet stage bij MMM, najaar 2004…

# Inhoud stage

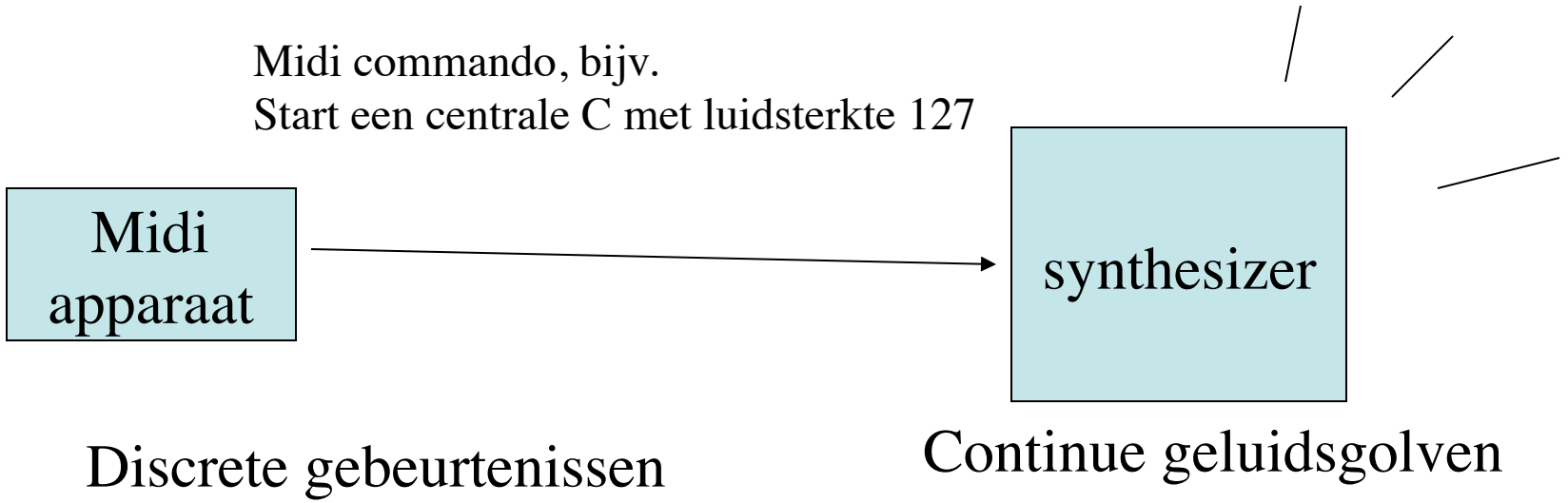
- Compositie mbv computer…
- Geschiedenis: verschillende benaderingen. Voornamelijke opdeling:

continue (signal processing)

vs.

discreet (symbolisch)

Opsplitsing duidelijk in midi:
discrete gebeurtenissen sturen
continue signalen aan.



# Voorbeeld midi sequence

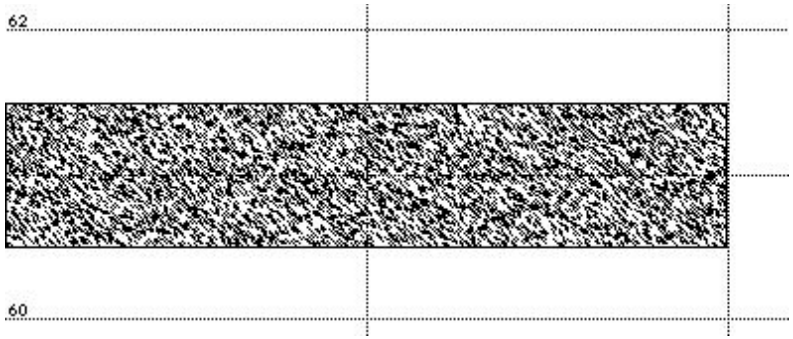
```
("00:00:00.000   0/0    ProgChange (00)")
("00:00:00.500   0/0    KeyOn (60 127)")
("00:00:01.500   0/0    KeyOff (60 127)")
```

- Ook mogelijk om continue verloop binnen een discrete gebeurtenissen te besturen, bijvoorbeeld vibrato of panning.

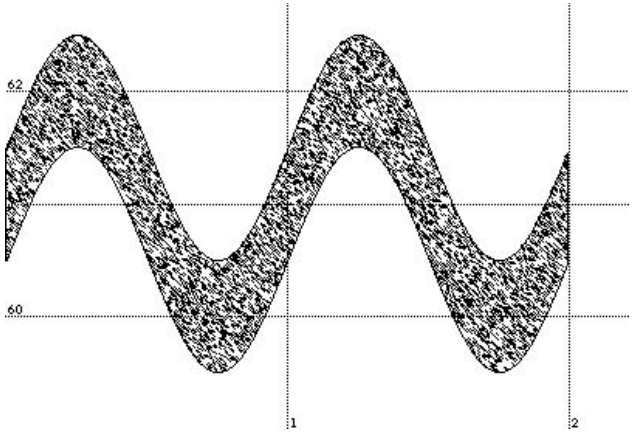
Continue verloop op te vatten als functie van tijd. Bijvoorbeeld:

constant(t) = 61

vibrato(t) = 61 + sin(Pi*t).




Constant(t)



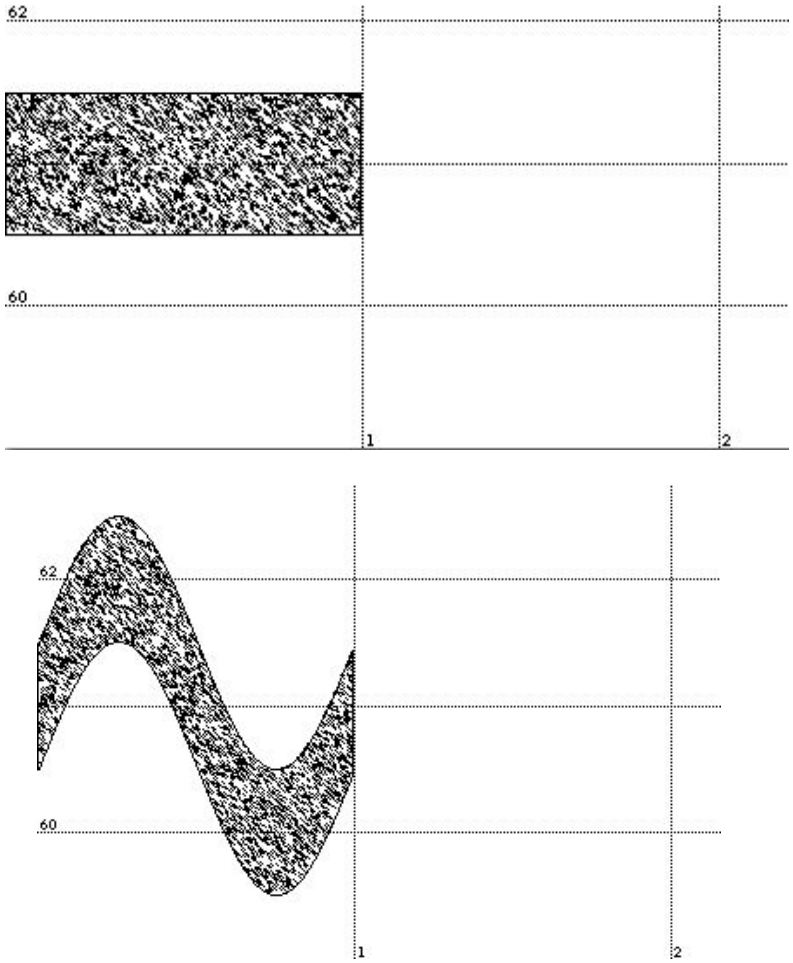

Vibrato(t)

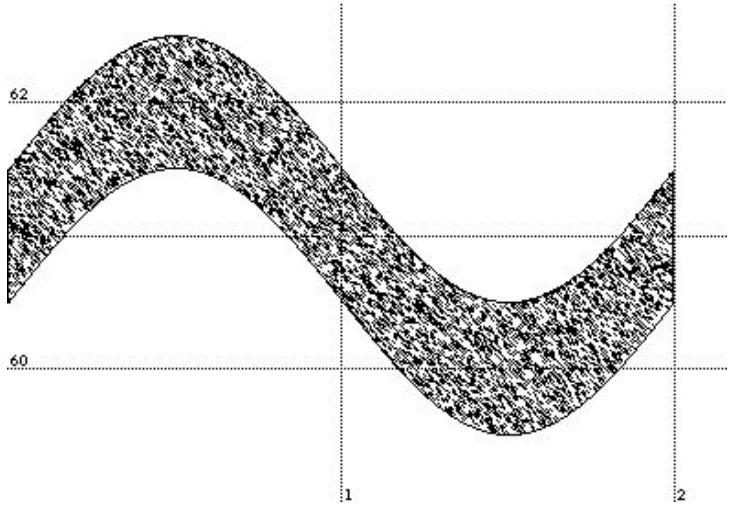
Probleem:

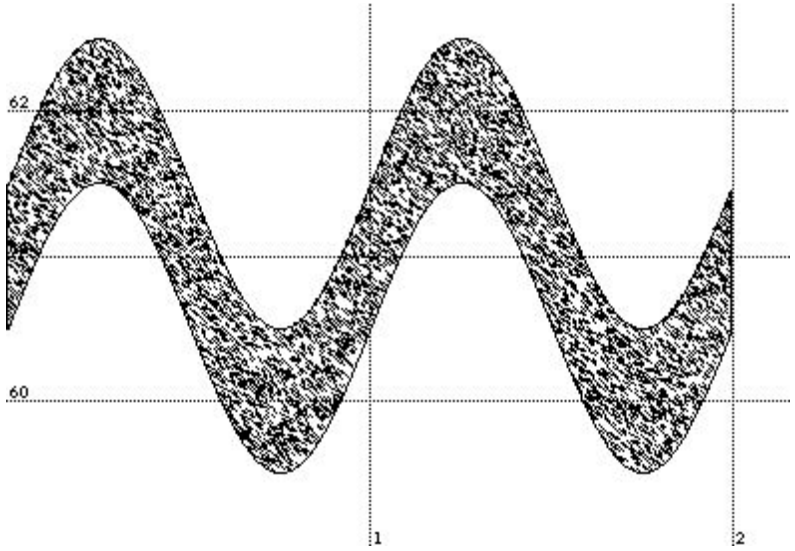
*hoe moeten de continue besturingsfuncties zich gedragen als de gebeurtenis waarop ze betrekking hebben een transformatie ondergaat?*

- 3 belangrijke voorbeelden: vibrato, glissando, ornament

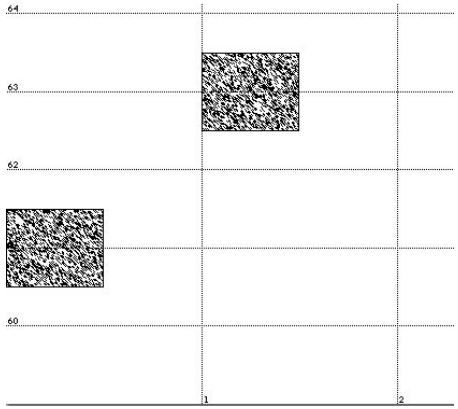
# Voorbeeld 1: vibrato en transformatie

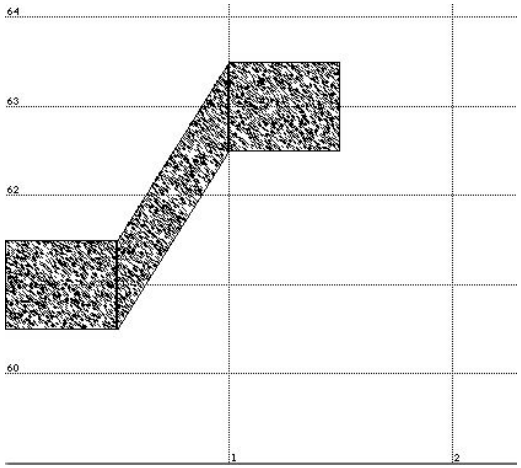


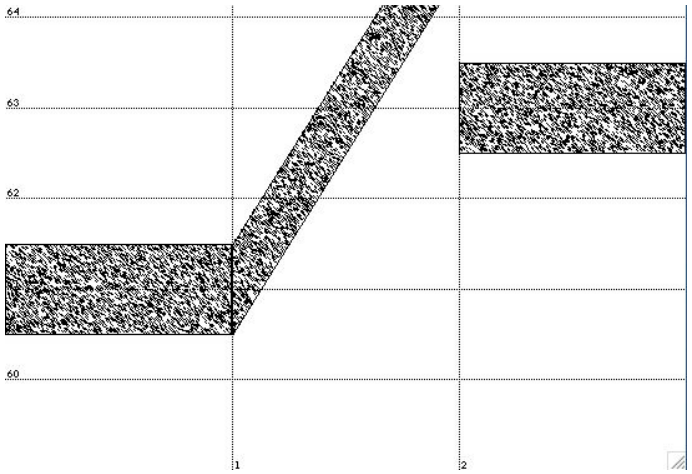

62
60
1
2

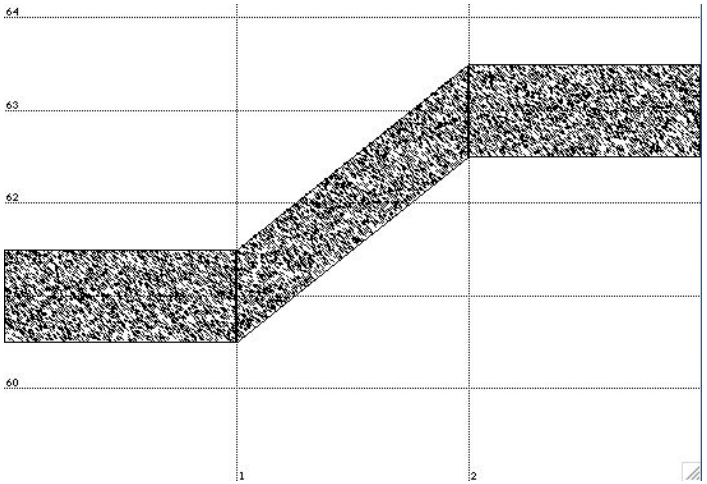

62
60
1
2

# Voorbeeld 2: glissando

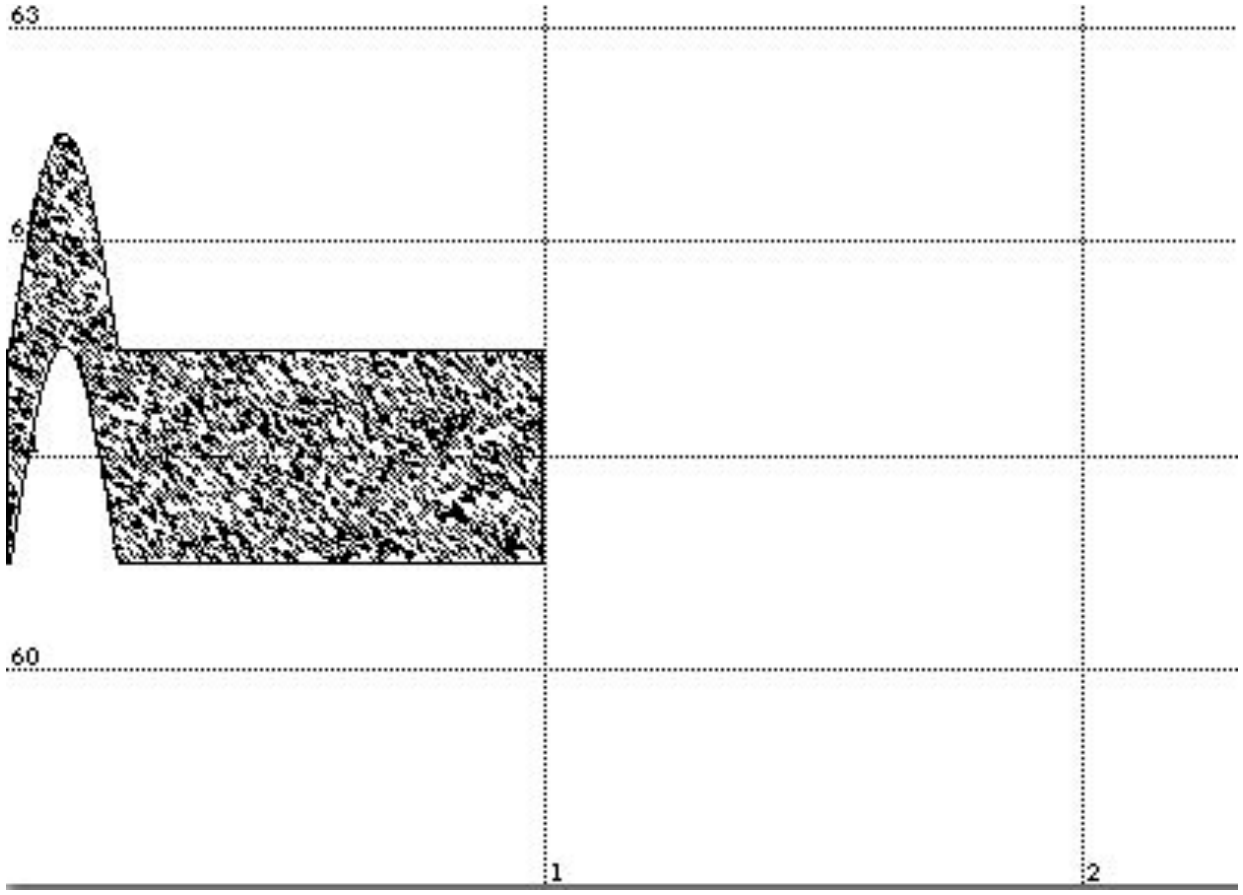


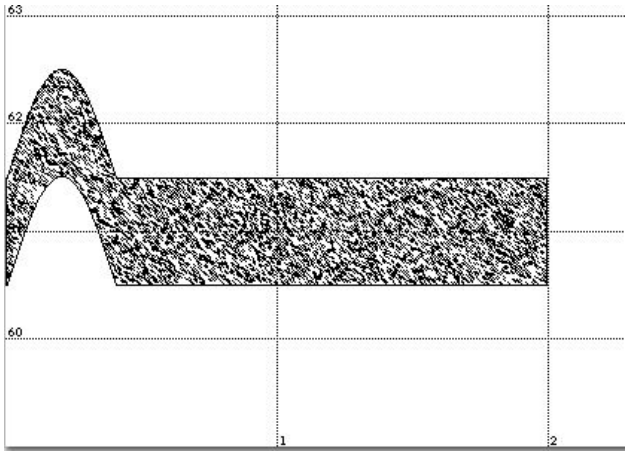



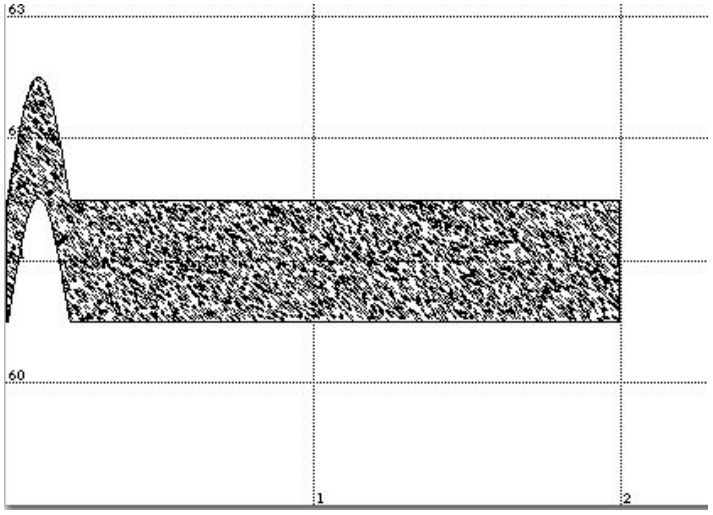

64
63
62
60
1
2


64
63
62
60
1
2

# Voorbeeld 3: ornament




63
62
60
1
2


63
60
1
2

- Oplossing: functies van meerdere soorten tijd
- Soorten tijd: starttijd van de discrete gebeurtenis, duratie van een discrete gebeurtenis, de actuele tijd tijdens het doorlopen van de discrete gebeurtenis

*Functies van deze soorten tijd kunnen het gedrag van bovenstaande voorbeelden op de gewenste manier representeren.*

# Stageomschrijving

*Maak een "legodoos" met bouwblokjes waarmee je van simpele GTF's complexere kunt bouwen*

GTF = Generalized Time Function: functies van een veralgemeend tijdsconcept.

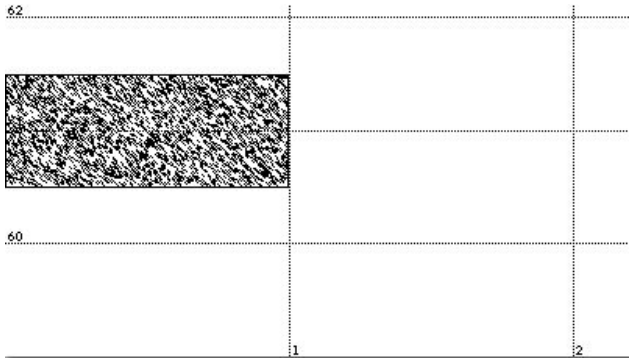
# Voorbeeld: concatenate

Conceptueel: “glissando van toonhoogte 61 naar 62 over een tijdsduur 1”

# Voorbeeld: concatenate

Interne representatie:

```
(note pitch duration amplitude)
(note 61 1 1) =>
```

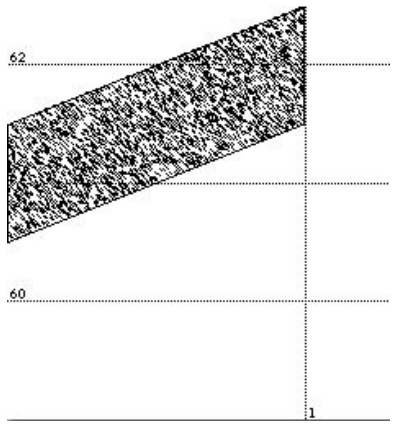


# Voorbeeld: concatenate

Interne representatie:

```
(note pitch duration amplitude)
(note (ramp 61 62) 1 1) =>
```

GTF



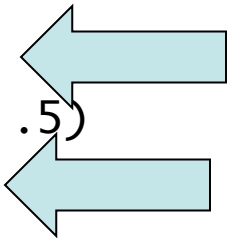
# Voorbeeld: concatenate

Conceptueel: “twee glissando’s aan elkaar vastgeplakt, van 61 naar 62 en terug”

# Voorbeeld: concatenate

Interne representatie:

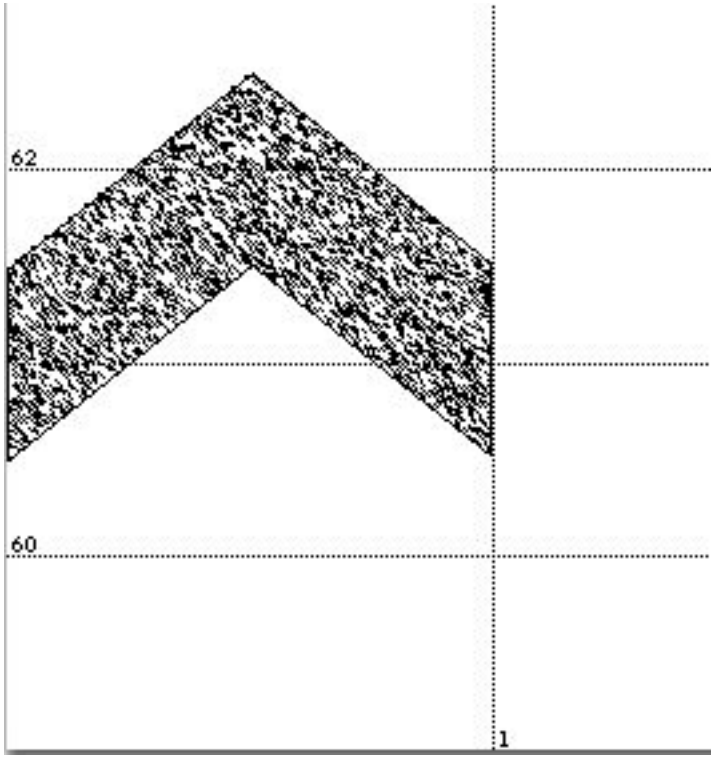
```
(draw (note (concat-gtf #'start-time
                        (ramp 61 62)
                       (proportional .5)
                        (ramp 62 61)
                        #'end-time
                        )
            1 1))
```



GTF's

# Voorbeeld: concatenate

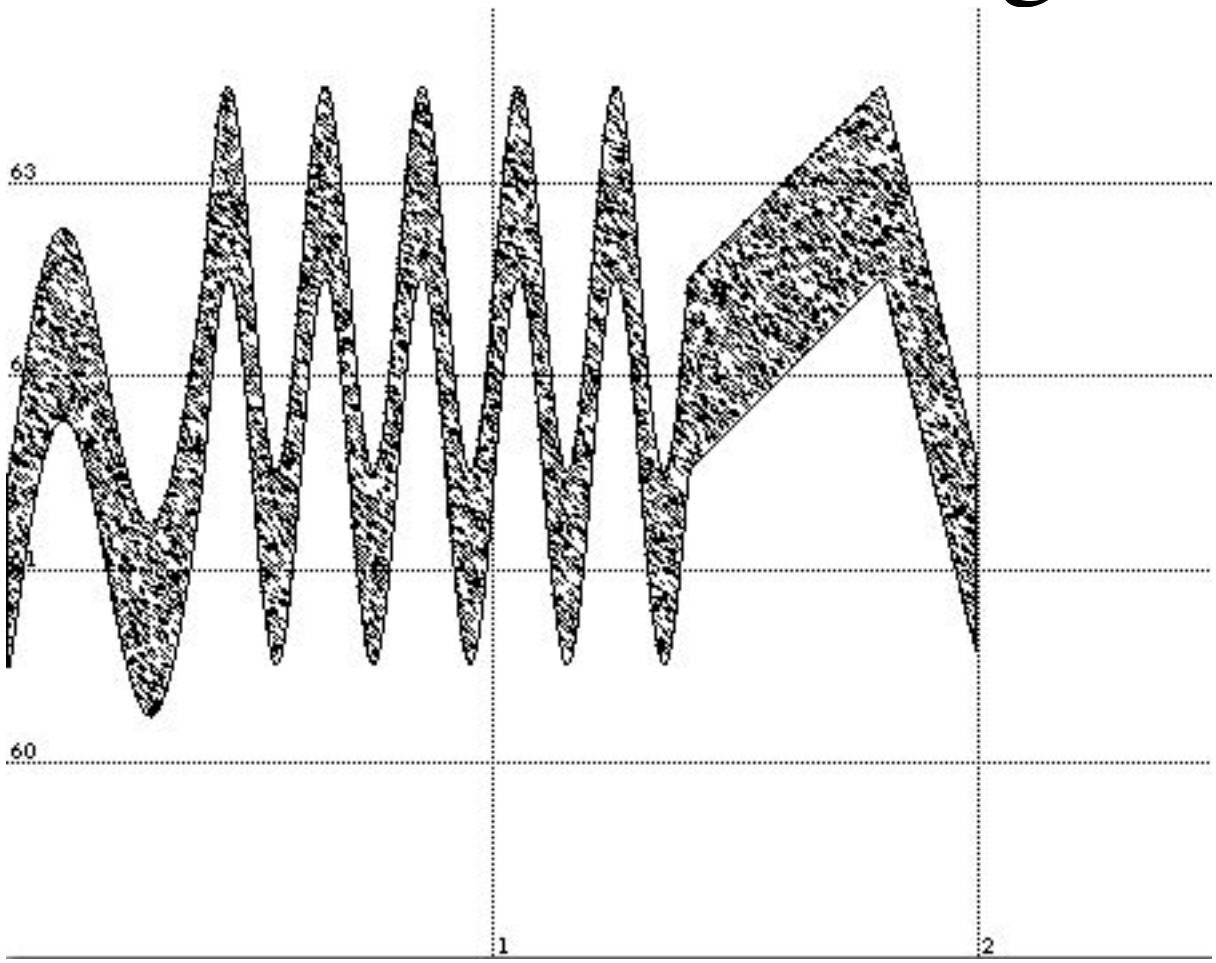
Grafisch:



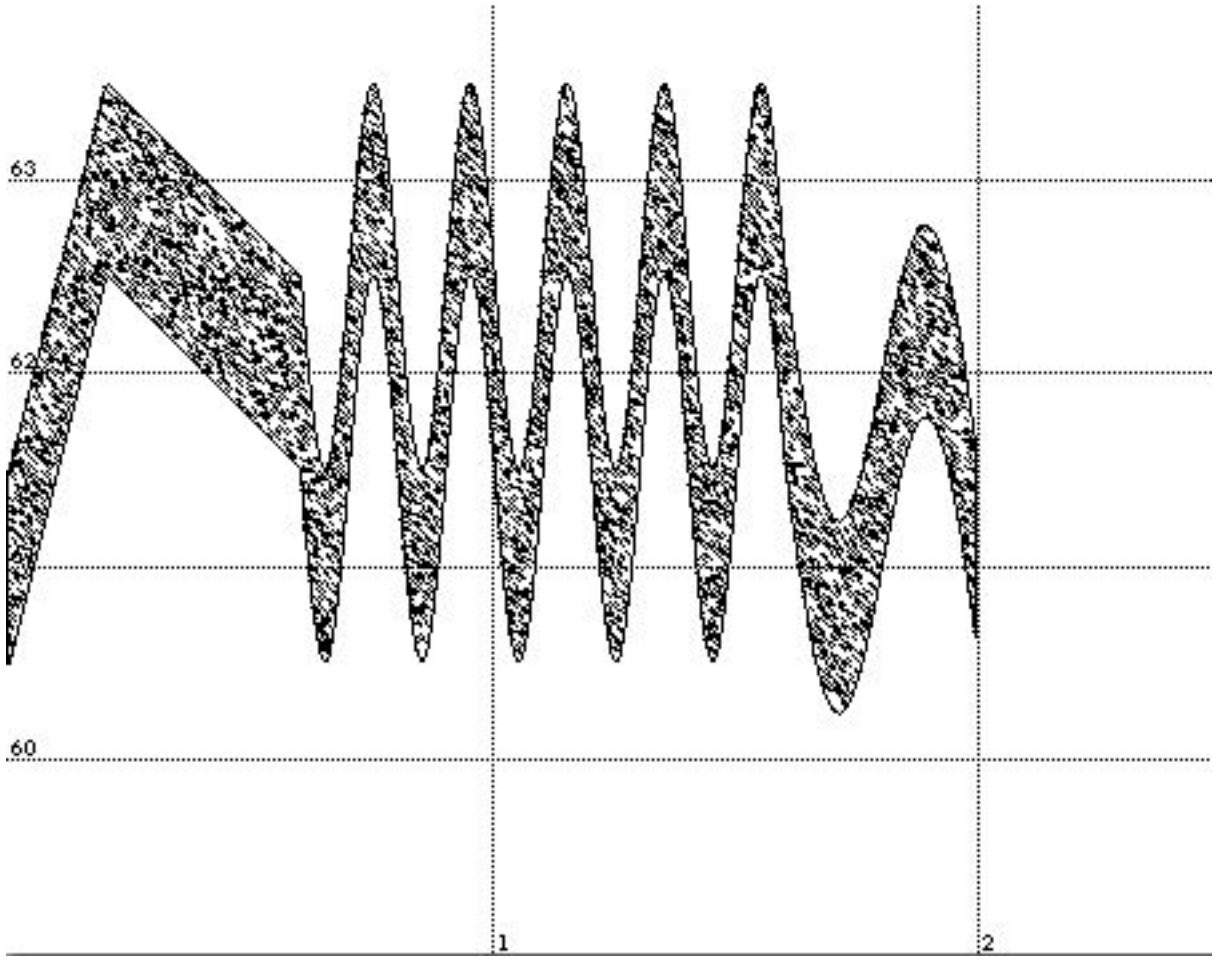
# Voorbeeld: reverse-gtf

```
(setf crazy-gtf (concat-gtf #'start-time
                            (gtf+ (lift-relative (sine .4) 0) (ramp 61
   62))
                            (proportional .2)
                            (gtf+ 62 (lift-relative (sine .2) 0))
                            (proportional .7)
                            (ramp 62 63)
                            (proportional .9)
                            (ramp 63 61)
                            #'end-time))
```

# Voorbeeld: reverse-gtf



# Voorbeeld: reverse-gtf



# Iets over Lisp

Waarom gebruik ik Lisp?

- Eigenschap van een functionele programmeertaal: functies kunnen andere functies opleveren: hogere orde functies
- Deze mogelijkheid niet aanwezig in: Java, C++, Pascal (Delphi)

# Lisp taal van de AI?

Mijn stelling: *"In Lisp kun je je ideeen sneller en mooier implementeren dan in andere talen."*

Waarom Lisp specifiek taal van de AI (wellicht)?

Lisp heeft zowel praktische (zie zometeen) als theoretische eigenschappen waardoor het een geschikte AI taal is.

Hedendaagse AI en Cognitiewetenschap: *intelligent gedrag is het resultaat van symboolmanipulatie*. Deze stelling is bekend onder de namen *Physical Symbol System Hypothesis* (PSSH) of *Representational Theory of Mind*.

Vereenvoudigd komt de PSSH (Newell, 1980) hierop neer:

1. Primitieve mentale concepten worden gerepresenteerd in een intelligent systeem door fysieke symbolen.
2. Complexe mentale concepten worden gerepresenteerd als structuren van fysieke symbolen.

PSSH (Newell, 1980) vervolg:

3. Intelligentie (een verzameling intelligente functies) is het resultaat van de manipulatie van structuren van fysieke symbolen. Die manipulatie gebeurt door programma's.
4. Programma's kunnen gerepresenteerd worden als fysieke symbool structuren. Op die manier kan een intelligent systeem *leren* (door de manipulatie van symboolstructuren die programma's representeren).

- De primitieve datastructuren van LISP: symbolen en lijsten
- Compositie van complexe structuren makkelijk en mooi te doen in Lisp
- Samengestelde functietoepassing makkelijk en mooi te doen in Lisp
- Programma en data worden opdezelfde manier gerepresenteerd: de weg geopend naar programma's die programma's kunnen modificeren en construeren

# Lisp voorbeeld praktijk

Faculteitsfunctie fac(n).

fac(1) = fac(0) = 1 (definitie)

fac(2) = 2 * 1 (= 2)

fac(3) = 3 * 2 * 1 (= 6)

fac(4) = 4 * 3 * 2 * 1 (= 24)

# Lisp voorbeeld praktijk

Faculteitsfunctie fac(n).

fac(1) = fac(0) = 1 (definitie)

fac(2) = 2 * fac(1) (= 2)

fac(3) = 3 * fac(2) (= 6)

fac(4) = 4 * fac(3) (= 24)

# Lisp voorbeeld praktijk

Faculteitsfunctie fac(n). Algemene definitie:

fac(1) = fac(0) = 1

fac(n) = n * fac(n-1), voor n > 1

Dit wordt ook wel een *recursieve* definitie genoemd.

# Intermezzo: prefix en infix notatie.

infix notatie:

3 + 5

4 < 7

prefix notatie (gebruikt in Lisp):

+ 3 5

< 4 7

# Lisp voorbeeld praktijk

Faculteitsfunctie fac(n). Algemene definitie:

fac(1) = fac(0) = 1

fac(n) = n * fac(n-1), voor n > 1

Idee uitgewerkt in Lisp:

```
(defun fac (n)
   (if (< n 2)
       1
       (* n (fac (- n 1)))))
```

# Lisp voorbeeld praktijk

Faculteitsfunctie fac(n). Algemene definitie:

fac(1) = fac(0) = 1

fac(n) = n * fac(n-1), voor n > 1

Uitproberen levert op:

> (fac 4)

24

# Lisp voorbeeld praktijk

## Nu hetzelfde in Java:

```
import java.io.*;

class Faculteit {

  public static int fac(int n) {
    if (n > 0)
      return n * fac(n - 1);
    else
      return 1;
  }


  public static void main(String[] args) throws IOException {
    BufferedReader toetsenbord;
    toetsenbord = new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Getal n ?");
    String invoertekst;
    invoertekst = toetsenbord.readLine();
    int getal;
    getal = Integer.parseInt(invoertekst);
    System.out.println(getal + "! = " + fac(getal));
  }

}
```

# Lisp voorbeeld praktijk

Om nu je programma te testen moet je eerst het geheel compileren, wat er op mijn computer als volgt uitzag:

*fsw20262:~/java borkent$* javac Faculteit.java

Daarna kun je eindelijk je programma gaan uitproberen:

*fsw20262:~/java borkent$* java Faculteit
Getal n ?
4
4! = 24

# Lisp

- Lisp is interactief, implementatie valt snel uit te proberen en te wijzigen
- (Daardoor) overhead voor zaken als in- en uitvoer naar het scherm is niet groot (itt tot het Java voorbeeld)
- Lisp is niet alleen functioneel, maar combineert belangrijke eigenschappen van verschillende talen

# Conclusies

- Combineren interesses met studiekeuze is mogelijk, zeker als studiekeuze = AI
- Computer is een handige tool bij muziekcompositie. Voorbeeld is mijn stageonderwerp.
- Lisp is een interactieve taal, waarin mogelijk is om snel en mooi je ideeen om te zetten in werkende code