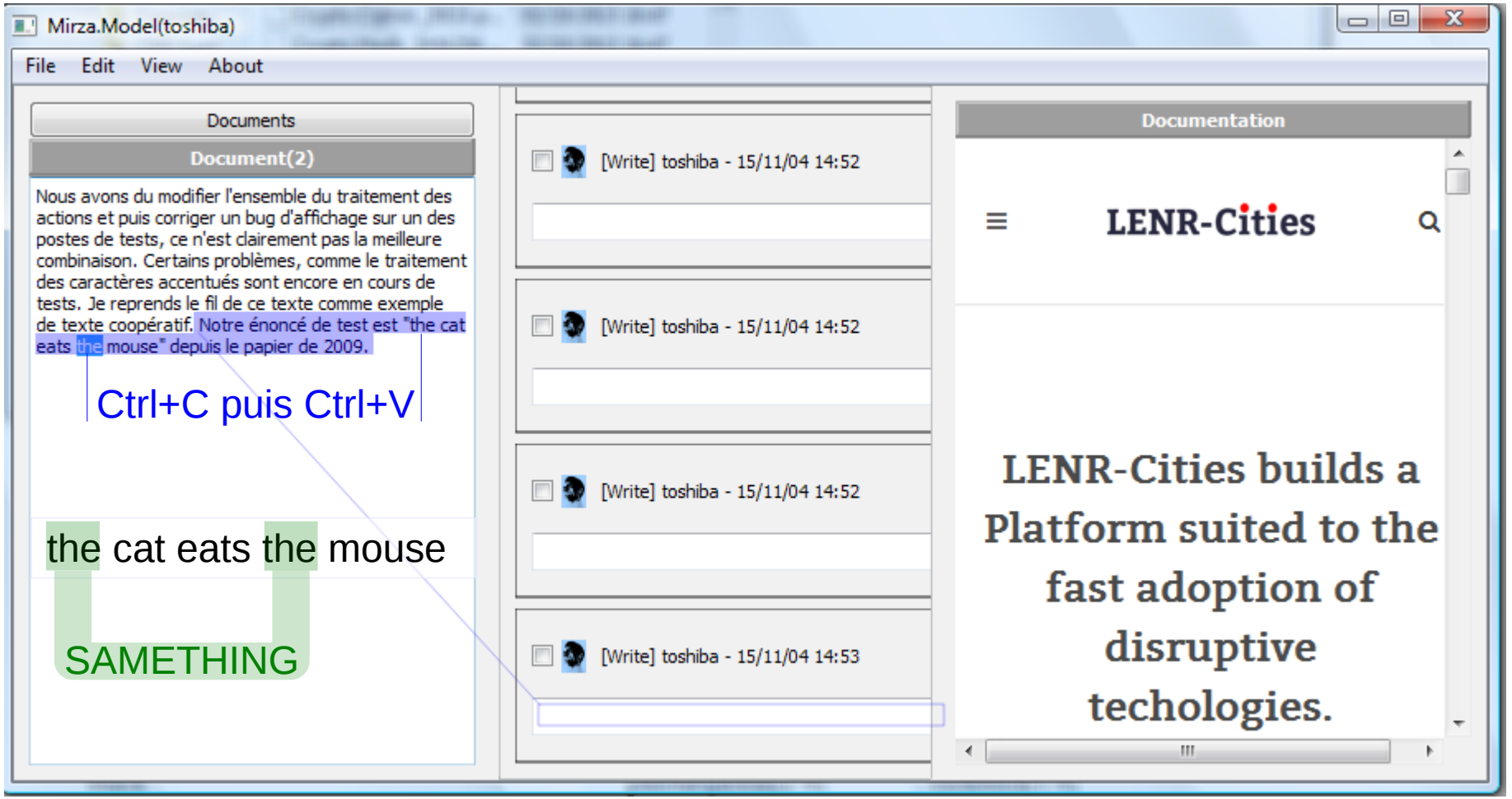
Clicker c'est programmer

Une approche cohérente pour la qualité de la coopération scientifique sur le réseau

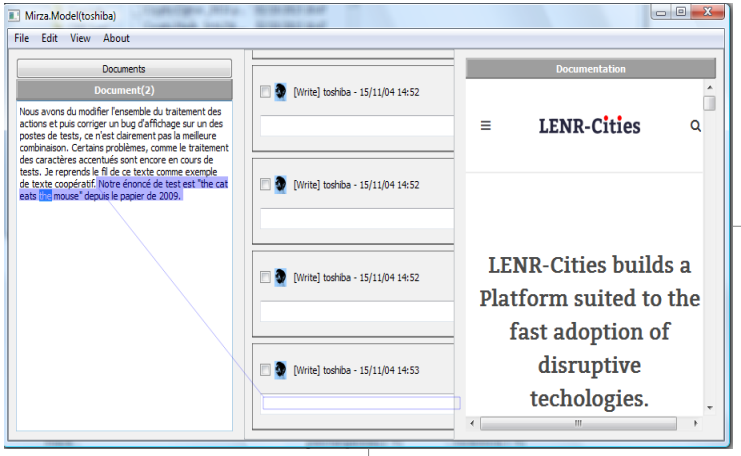
Deux énoncés semblables



Ex. de développement : Le « *the* » de « *cat* » est le **même** que le « *the* » de « *mouse* »

→ **Copier « *the* » sur « *the* »**

Clicker c'est programmer



x = copy(int, int)
ou
x = input()
puis
write(x, int, int)

```
Python 2.7.8 (Console)
[...]
>>> x = input()
the cat eats the mouse.
>>> write(x, 0, 0)
>>> the = copy(0,3)
 SAMETHING
>>> write(the, 13, 16)
>>> _
the cat eats the mouse.
```

```
_ = Controller()
def write(x, a, b):
    _<=_[:a] + x + _[b:]
def copy(a, b):
    return _[a:b]
def input():
    return raw_input()
```

```
class Co
    def _
    def _
    def _
    def _
```

S

A

M

O

Les solutions existantes

- *Simples* : basées sur « *type de base* » (str)
- Pas de mémoire du geste de *glisser-déposer*

S

A

M

O

©

```
Python 2.7.8 (Console)
>>> x = input()
the cat eats the mouse.
>>> write(x, 0, 0)
>>> the = copy(0,3)

>>> write(the, 13, 16)
>>> _
the cat eats the mouse.
```

```
_ = View()
def write(x, a, b):
    _ <= _[:a] + x + _[b:]
def copy(a, b):
    return _[a:b]
def input():
    return raw_input()
```

```
class Controller(object):
    def __init__(self):
        self.model = str()
    def __le__(self, _):
        self.model = _
    def __getitem__(self, _):
        return self.model[_.start: _.stop]
    def __repr__(self):
        return repr(self.model)
```

Comment obtenir ?

- Un système qui mémorise...

Drag 'n drop <u>means</u> « *the samething* »!

```
Python 2.7.8 (Console)
>>> x = input()
the cat eats the mouse.
>>> write(x, 0, 0)
>>> the = copy(0,3)
 SAMETHING
>>> write(the, 13, 16)
>>> _
the cat eats the mouse.
```

```
_ = View()
def write(x, a, b):
    _ <= _[:a] + x + _[b:]
def copy(a, b):
    return _[a:b]
def input():
    return raw_input()
```

```
[…]
class Controller(View):
        def __init__(self):
                [...]
        def __le__(self, _):
                [...]
        def __getitem__(self, _):
                [...]
        def __repr__(self):
                [...]
```

Un ancien problème (1965)

- Formulé par Ted Nelson, inventeur du mot « *hypertext* »
  - Xanadu, plate-forme à « *transclusion* »
- Repris par Jaron Lanier, Personal Democracy 2012
  - « *Comment un geek va (failli) sauver l'économie mondiale* »
- Voie d'attaque :
  - Caractériser l'objet échangé : « the »
    - « *hyperlien à double sens* » : syntaxe (1965 ?)
  - TCP sockets more or less (2011)...
  - « *Uniform Resource Locators* » :
    - Tim Berners-Lee – 1994
    - « *hyperlien* » => no one know who knows it.
  - 2015 : presque l'interface *write-copy-input*
- 19 Juin 2012 : J'ai l'algèbre pour le faire !
- **Novembre 2015** : j'ai le modèle pour le faire !

Pierre Gradit a posté le **19 juin 2012** à 09h40
Tout pareil, j'ai construit depuis 2008 une théorie ("algébrique") capable de gérer des "**hyperliens à double sens**" et dont les conséquences pratiques ("informatiques") sont la suppression de la duplication des données (donc sans les "fichiers" tels que nous les connaissons - obstacle conceptuel franchi en 2009) et la propriété intellectuelle individuelle absolue des données et des procédés avec exactement le même type de "facturation en cascade". Pour valoriser cette construction, j'ai crée une société en 2008, breveté le principale conséquence brevetable de la théorie qui est le mécanisme capable de remplacer le transport de fichier en 2010, maquetté le mécanisme avec des dizaines de démonstrations en 2011. Tout le monde s'en fout : c'est un plaisir rare d'avoir des idées qui changent le monde, rare mais pénible, surtout au début.

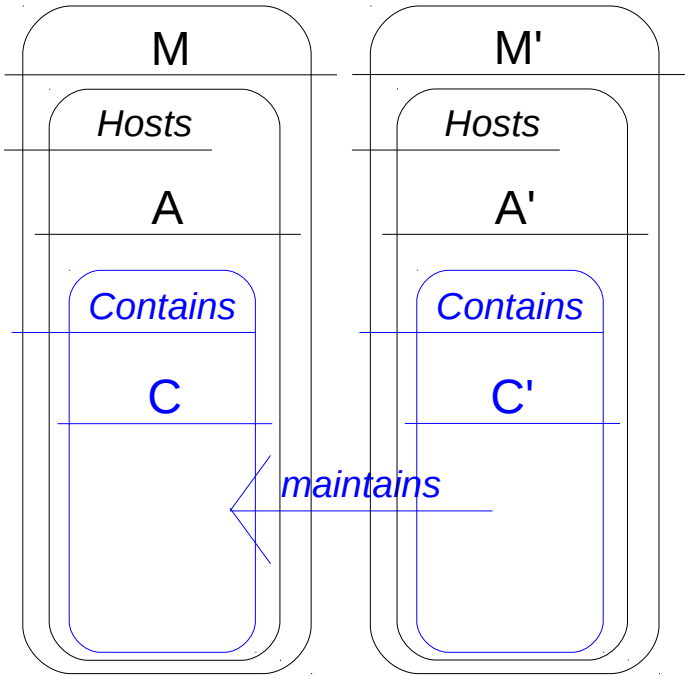
```
Python 2.7.8 (Console)
>>> x = input()
the cat eats the mouse.
>>> write(x, 0, 0)
>>> the = copy(0,3)
    BD-LINK
>>> write(the, 13, 16)
>>> _
the cat eats the mouse.
```

Expérience de la solution

- **Programme de R&D (2010-2011) :**
  - **Mirza** [2011] : 90 k€ (64k€ de love-money, 27k€ OSEO)
  - **Etude de marché** : « les tableurs ne sont pas la voie d'accès au marché »
  - « *Method for partial learning sharing of a software appliocation* » (24843 – France, UE, US) 06/12/2010
  - **Maquette** : Calcul d'un devis par la méthode des partages partiels (2011)
- Survie d'une société (2008-...) : SARL MEZZONOMY [504 641 473]
  - Objet social = « ***Logiciels programmables pour l'ingénieur*** »
  - En charge depuis 2008 de « *valoriser une algèbre graphique* »
- **CARGO** : Machine industrielle de génération d'« *outils métiers* »
  - depuis **2011** : 10k€, **2012** : 2 k€, **2013** : 25k€, **2014** : 80k€, **2015** : 11k€
    - 2015 : QAO (Qualité Assisté par Ordinateur – i3L),
    - 2015 : QDF (Qualité Données Fournisseurs – SAFRAN)
  - Les « *outils-métiers* » sont des feuillet qui ont « *réussi* »
  - Améliorer la gestion de configuration des « *réussites* »
  - Existe aussi en CARGOWEB (consolidation planifiée)

Le brevet 24843

- Procédé de « *partage partiel d'application* » :
  - Depuis une application A, exécutée sur une machine M, utilisée par U
  - Vers une application A', exécutée sur une machine M', utilisée par U'
- Quatre phases :
  - Utilisation de A par U,
  - Définition par U d'un domaine d'action C < A
  - Exportation par U de D sous forme d'une adresse
  - Reproduction sur M' d'un reflet interactif C' de C
- Propriété du reflet :
  - C' présente un « *comportement identique* » à C
  - Comportement : « *Apparences = Calcul(Actions)* »
  - Les actions sur C' sont stockées sur M !

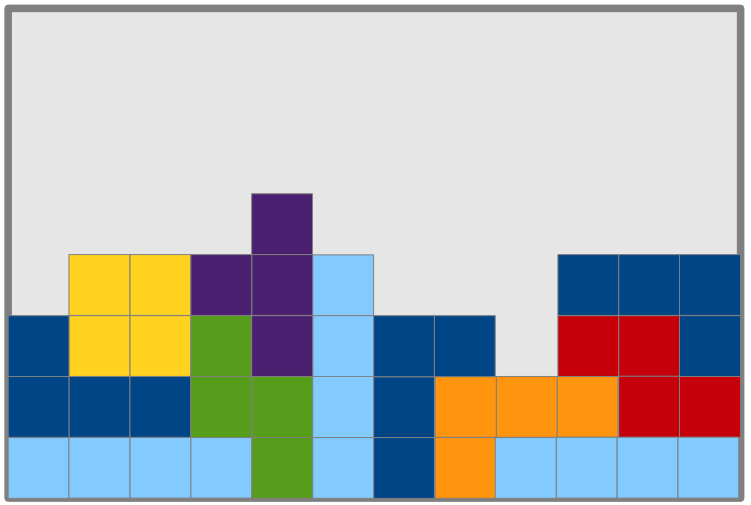


Apparence = Calcul(Actions)

- Apparence d'une partie de Tetris
  - Calcul = « *vue du dessus* »

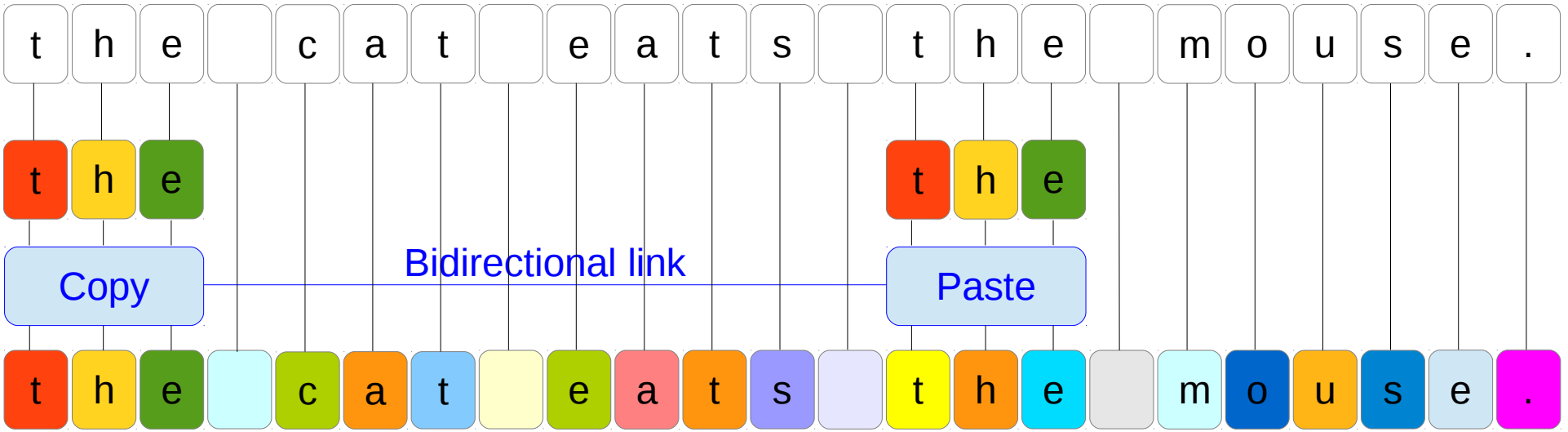
Apparence =

Actions =



Exposition simplifiée

- Les deux articles sont identiques, et non seulement de même apparence.

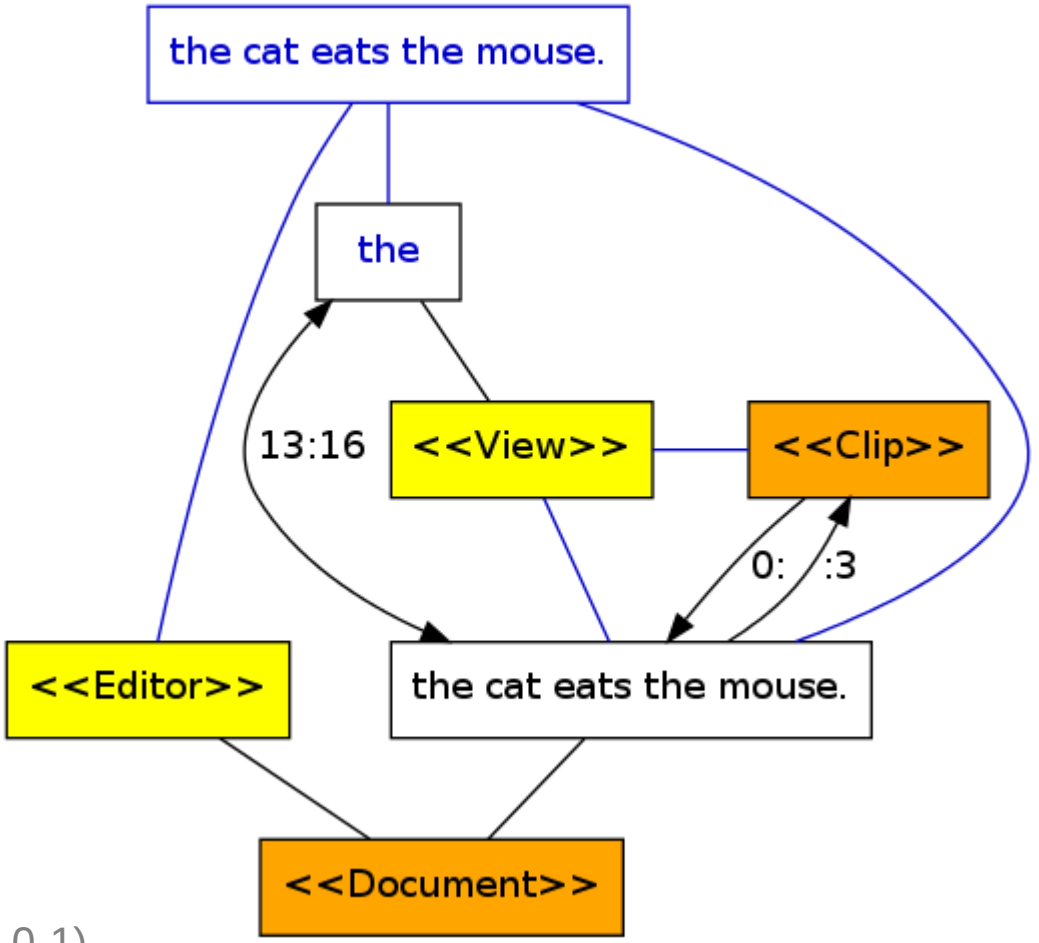



```
python -m mirza.view empty
>>> x = raw_input()
the cat eats the mouse.
>>> gradit_4.write(x, 0, 0)
>>> the = gradit_4.copy(0,3)
>>> gradit_4.write(the, 13, 16)
```

```
gradit  = __('gradit')
mzz = ___('http://mezzonomy.com')
100., gradit,  1, mzz.Document()
149., gradit,  2, mzz.Text("", gradit-1)
255., gradit,  3, mzz.Editor(gradit-1)
260., gradit,  4, mzz.Expose(gradit-3, _gradit-2)
```

Test unitaire

```
$ python -m mirza.view empty
151106:104412 WARNING  'empty' is the repository
Python 2.7.8 (InteractiveConsole)
>>> dump()
_0 = __('gradit')
_1 = ___(http://mezzonomy.com)
100.0, _0, 1, _1.Document()
255.0, _0, 3, _1.Editor(_0-1)
260.0, _0, 4, _1.Expose(_0-3,_0-2)
>>> x = raw_input()
the cat eats the mouse.
>>> gradit_4.write(x, 0, 0)
>>> the = gradit_4.clip(0,3)
>>> gradit_4.write(the, 13, 16)
>>> dump()
_0 = __('gradit')
_1 = ___(http://mezzonomy.com)
100.0000, _0, 1, _1.Document()
255.0000, _0, 3, _1.Editor(_0-1)
391.9503, _0, 5, _1.Write(u'the cat eats the mouse.',_0-1)
394.7824, _0, 6, _1.Clip((_0-5)[0:],(_0-5)[:3])
397.3101, _0, 7, _1.SubExpose(_0-6,_0-5)
397.3103, _0, 8, _1.Write(_0-7,(_0-5)[13:16])
420.3764, _0, 4, _1.Expose(_0-3,_0-2,_0-5,_0-7)
>>> dot()
```



Conception générale

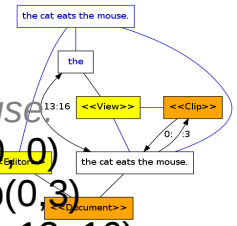
Bank

```
_0 = __('gradit')
_1 = ___(http://mezzonomy.com)
100.0000, _0, 1, _1.Document()
255.0000, _0, 3, _1.Editor(_0-1)
391.9503, _0, 5, _1.Write(u'the cat eats the mouse.',_0-1)
394.7824, _0, 6, _1.Clip((_0-5)[0:],(_0-5)[:3])
397.3101, _0, 7, _1.SubExpose(_0-6,_0-5)
397.3103, _0, 8, _1.Write(_0-7,(_0-5)[13:16])
420.3764, _0, 4, _1.Expose(_0-3,_0-2,_0-5,_0-7)
```

Model

View

```
> x = raw_input()
the cat eats the mouse
> gradit_4.write(x, 0, 0)
> the = gradit_4.clip(0,3)
> gradit_4.write(the, 13, 16)
> dot()
```



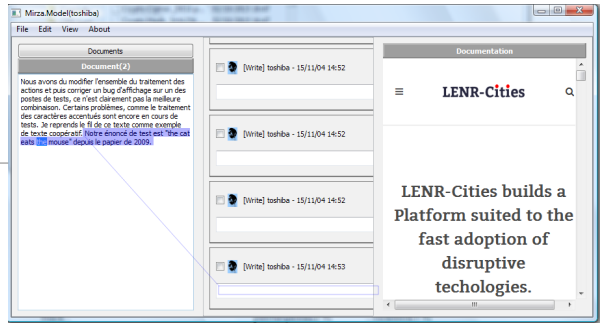
Cache

(gradit_7)
(gradit_4)
(gradit_5)
(gradit_2)
(gradit)

Identification

Log

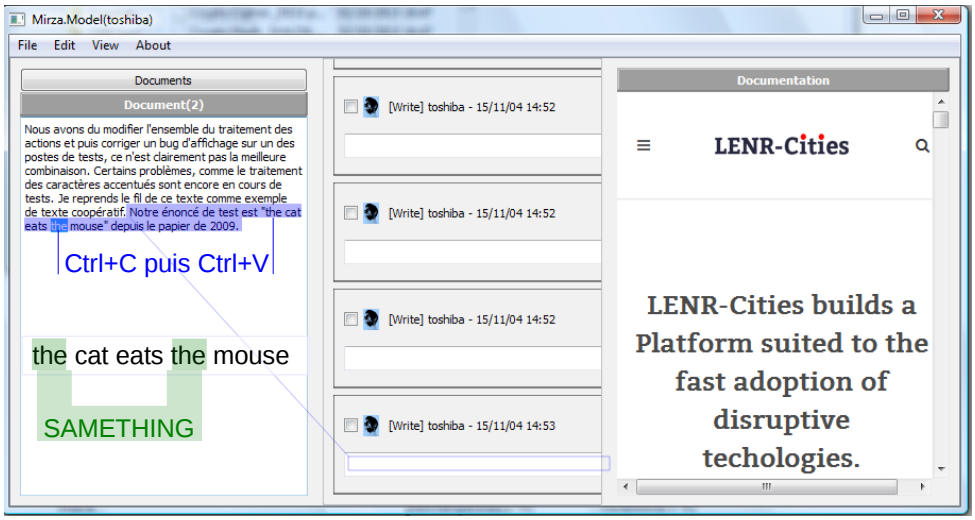
Controller



Clicker c'est programmer

Une approche cohérente pour la qualité de la coopération scientifique sur le réseau

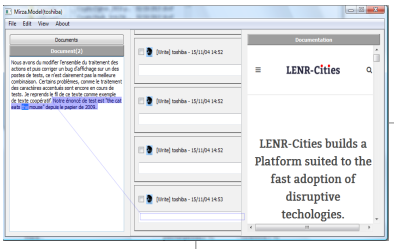
Deux énoncés semblables



Ex. de développement : Le « *the* » de « *cat* » est le **même** que le « *the* » de « *mouse* »

→Copier « *the* » sur « *the* »

Clicker c'est programmer



x = copy(int, int)
ou
x = input()
puis
write(x, int, int)

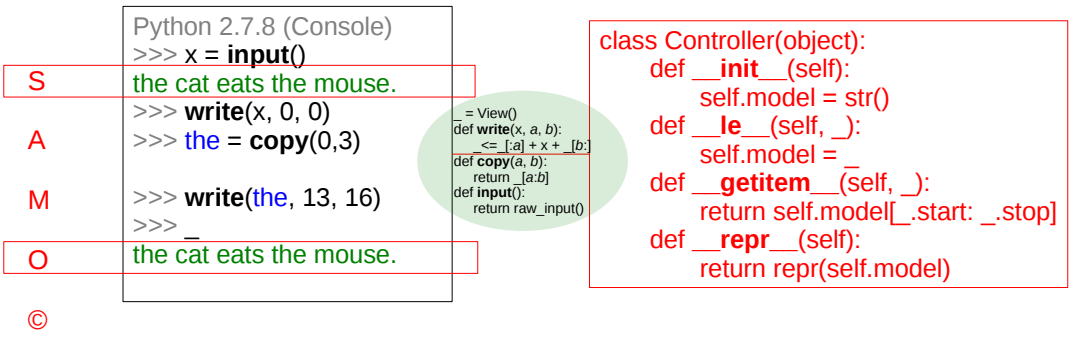
```
Python 2.7.8 (Console)
[...]
>>> x = input()
S       the cat eats the mouse.
>>> write(x, 0, 0)
>>> the = copy(0,3)
A       SAMETHING
>>> write(the, 13, 16)
M
>>> _
O       the cat eats the mouse.
```

```
_ = Controller()
def write(x, a, b):
    _<=_[:a] + x + _[b:]
def copy(a, b):
    return _[a:b]
def input():
    return raw_input()
```

```
class Controller(object):
    def __init__(self):
        self.model = str()
    def __le__(self, _):
        self.model = _
    def __getitem__(self, _):
        return self.model[_.s
    def __repr__(self):
        return repr(self.mode
```

Les solutions existantes

- *Simples* : basées sur « *type de base* » (str)
- Pas de mémoire du geste de *glisser-déposer*



Comment obtenir ?

- Un système qui mémorise...

  Drag 'n drop <u>means</u> « *the samething* »!

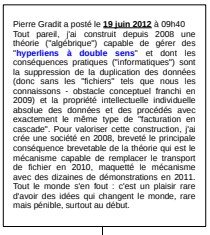
```
Python 2.7.8 (Console)
>>> x = input()
the cat eats the mouse.
>>> write(x, 0, 0)
>>> the = copy(0,3)
 SAMETHING
>>> write(the, 13, 16)
>>> _
the cat eats the mouse.
```

```
_ = View()
def write(x, a, b):
    x[a:] = x[:a] + x + x[b:]
def copy(a, b):
    return _[a:b]
def input():
    return raw_input()
```

```
[…]
class Controller(View):
        def __init__(self):
                [...]
        def __le__(self, _):
                [...]
        def __getitem__(self, _):
                [...]
        def __repr__(self):
                [...]
```

Un ancien problème (1965)

- Formulé par Ted Nelson, inventeur du mot « *hypertext* »
  - Xanadu, plate-forme à « *transclusion* »
- Repris par Jaron Lanier, Personal Democracy 2012
  - « *Comment un geek va (failli) sauver l'économie mondiale* »
- Voie d'attaque :
  - Caractériser l'objet échangé : « the »
    - « *hyperlien à double sens* » : syntaxe (1965 ?)
  - TCP sockets more or less (2011)...
  - « *Uniform Resource Locators* » :
    - Tim Berners-Lee – 1994
    - « *hyperlien* » => no one know who knows it.
  - 2015 : presque l'interface *write-copy-input*
- 19 Juin 2012 : J'ai l'algèbre pour le faire !
- **Novembre 2015** : j'ai le modèle pour le faire !


Pierre Gradit a posté le **19 juin 2012** à 09h40
Tout pareil, j'ai construit depuis 2008 une théorie ("algébrique") capable de gérer des "**hyperliens à double sens**" et dont les conséquences pratiques ("informatiques") sont la suppression de la duplication des données (donc sans les "fichiers" tels que nous les connaissons - obstacle conceptuel franchi en 2009) et la propriété intellectuelle individuelle absolue des données et des procédés avec exactement le même type de "facturation en cascade". Pour valoriser cette construction, j'ai crée une société en 2008, breveté le principale conséquence brevetable de la théorie qui est le mécanisme capable de remplacer le transport de fichier en 2010, maquetté le mécanisme avec des dizaines de démonstrations en 2011. Tout le monde s'en fout : c'est un plaisir rare d'avoir des idées qui changent le monde, rare mais pénible, surtout au début.

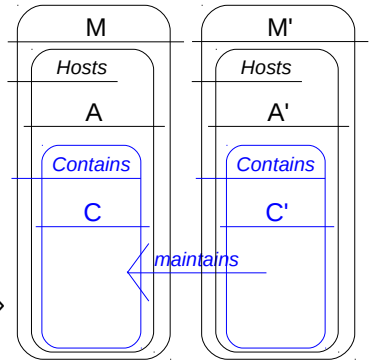
```
Python 2.7.8 (Console)
>>> x = input()
the cat eats the mouse.
>>> write(x, 0, 0)
>>> the = copy(0,3)
    BD-LINK
>>> write(the, 13, 16)
>>> _
the cat eats the mouse.
```

Expérience de la solution

- **Programme de R&D (2010-2011) :**
  - **Mirza** [2011] : 90 k€ (64k€ de love-money, 27k€ OSEO)
  - **Etude de marché** : « les tableurs ne sont pas la voie d'accès au marché »
  - « *Method for partial learning sharing of a software appliocation* » (24843 – France, UE, US) 06/12/2010
  - **Maquette** : Calcul d'un devis par la méthode des partages partiels (2011)
- Survie d'une société (2008-...) : SARL MEZZONOMY [504 641 473]
  - Objet social = « ***Logiciels programmables pour l'ingénieur*** »
  - En charge depuis 2008 de « *valoriser une algèbre graphique* »
- **CARGO** : Machine industrielle de génération d'« *outils métiers* »
  - depuis **2011** : 10k€, **2012** : 2 k€, **2013** : 25k€, **2014** : 80k€, **2015** : 11k€
    - 2015 : QAO (Qualité Assisté par Ordinateur – i3L),
    - 2015 : QDF (Qualité Données Fournisseurs – SAFRAN)
  - Les « *outils-métiers* » sont des feuillet qui ont « *réussi* »
  - Améliorer la gestion de configuration des « *réussites* »
  - Existe aussi en CARGOWEB (consolidation planifiée)

Le brevet 24843

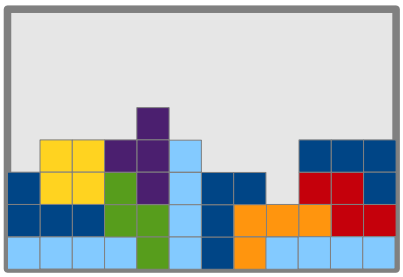
- Procédé de « *partage partiel d'application* » :
  - Depuis une application A, exécutée sur une machine M, utilisée par U
  - Vers une application A', exécutée sur une machine M', utilisée par U'
- Quatre phases :
  - Utilisation de A par U,
  - Définition par U d'un domaine d'action C < A
  - Exportation par U de D sous forme d'une adresse
  - Reproduction sur M' d'un reflet interactif C' de C
- Propriété du reflet :
  - C' présente un « *comportement identique* » à C
  - Comportement : « *Apparences = Calcul(Actions)* »
  - Les actions sur C' sont stockées sur M !



Apparence = Calcul(Actions)

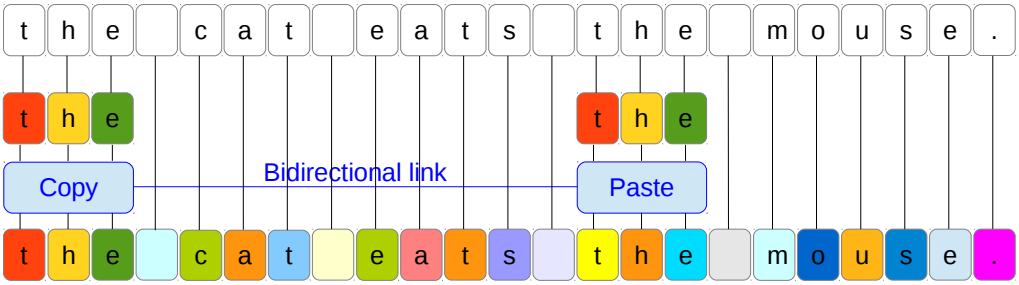
- Apparence d'une partie de Tetris
  - Calcul = « *vue du dessus* »

Apparence = 

Actions = 

Exposition simplifiée

- Les deux articles sont identiques, et non seulement de même apparence.

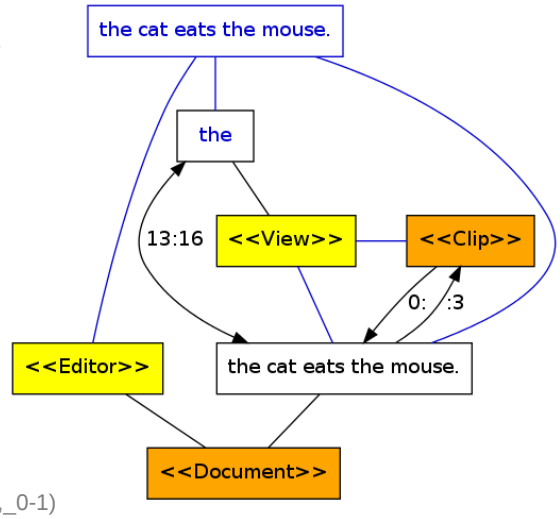



```
python -m mirza.view empty
>>> x = raw_input()
the cat eats the mouse.
>>> gradit_4.write(x, 0, 0)
>>> the = gradit_4.copy(0,3)
>>> gradit_4.write(the, 13, 16)
```

```
gradit  = __('gradit')
mzz = ___('http://mezzonomy.com')
100., gradit,  1, mzz.Document()
149., gradit,  2, mzz.Text("", gradit-1)
255., gradit,  3, mzz.Editor(gradit-1)
260., gradit,  4, mzz.Expose(gradit-3, _gradit-2)
```

Test unitaire

```
$ python -m mirza.view empty
151106:104412 WARNING  'empty' is the repository
Python 2.7.8 (InteractiveConsole)
>>> dump()
_0 = __('gradit')
_1 = ___(http://mezzonomy.com)
100.0, _0, 1, _1.Document()
255.0, _0, 3, _1.Editor(_0-1)
260.0, _0, 4, _1.Expose(_0-3,_0-2)
>>> x = raw_input()
the cat eats the mouse.
>>> gradit_4.write(x, 0, 0)
>>> the = gradit_4.clip(0,3)
>>> gradit_4.write(the, 13, 16)
>>> dump()
_0 = __('gradit')
_1 = ___(http://mezzonomy.com)
100.0000, _0, 1, _1.Document()
255.0000, _0, 3, _1.Editor(_0-1)
391.9503, _0, 5, _1.Write(u'the cat eats the mouse.',_0-1)
394.7824, _0, 6, _1.Clip((_0-5)[0:],(_0-5)[:3])
397.3101, _0, 7, _1.SubExpose(_0-6,_0-5)
397.3103, _0, 8, _1.Write(_0-7,(_0-5)[13:16])
420.3764, _0, 4, _1.Expose(_0-3,_0-2,_0-5,_0-7)
>>> dot()
```



Conception générale