Réalité Virtuelle

Animation

Evolution de l'état d'objets

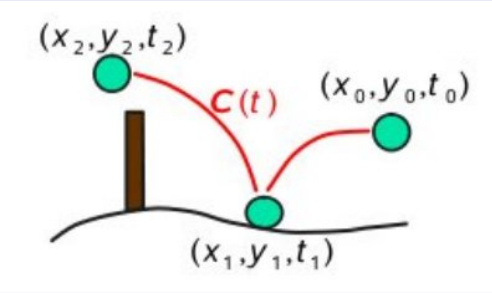
```
<Transform name="tux00" translation="10.0 0.0 0.0" >
  <Sphere rayon="1.0"/>
</Transform>
```

Modification du vecteur associé à l'attribut translation
de l'objet `tux00`

Interpolation

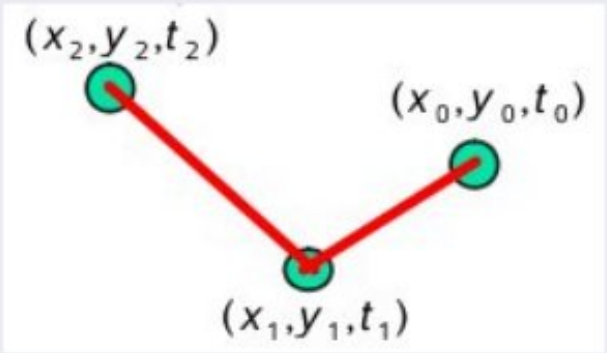
Interpolation

- de position
- d'orientation
- de couleur



- Données : $(x_i, y_i, t_i), i \in [0, n]$
- Résultat : $C(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}$ telle que $C(t_i) = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$

Interpolation linéaire



Problème simple

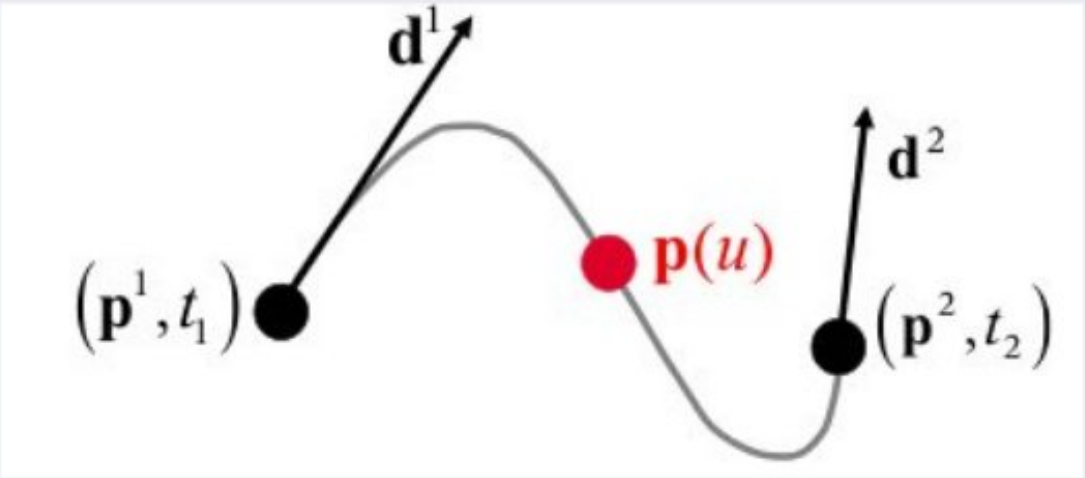
Interpolation linéaire entre deux points supposant que $t_0 = 0$ et $t_1 = 1$

$$x(t) = x_0(1 - t) + x_1 t$$

Description d'une courbe

$$x(t) = \begin{cases} \frac{t_1 - t}{t_1 - t_0} x_0 + \frac{t - t_0}{t_1 - t_0} x_1 & , \quad t \in [t_0, t_1[\\ \frac{t_2 - t}{t_2 - t_1} x_1 + \frac{t - t_1}{t_2 - t_1} x_2 & , \quad t \in [t_1, t_2] \end{cases}$$

Interpolation cubique



$$p(u) = h_1(u)d_1 + h_2(u)p_1 + h_3(u)p_2 + h_4(u)d_2 \text{ avec } u = \frac{t-t_1}{t_2-t_1}$$

$$\begin{cases} h_1(u) &= u^3 - 2u^2 + u \\ h_2(u) &= 2u^3 - 3u^2 + 1 \\ h_3(u) &= -2u^3 + 3u^2 \\ h_4(u) &= u^3 - u^2 \end{cases}$$

Modèle de steering

- Modélisation cinématique
- masse ponctuelle orientée

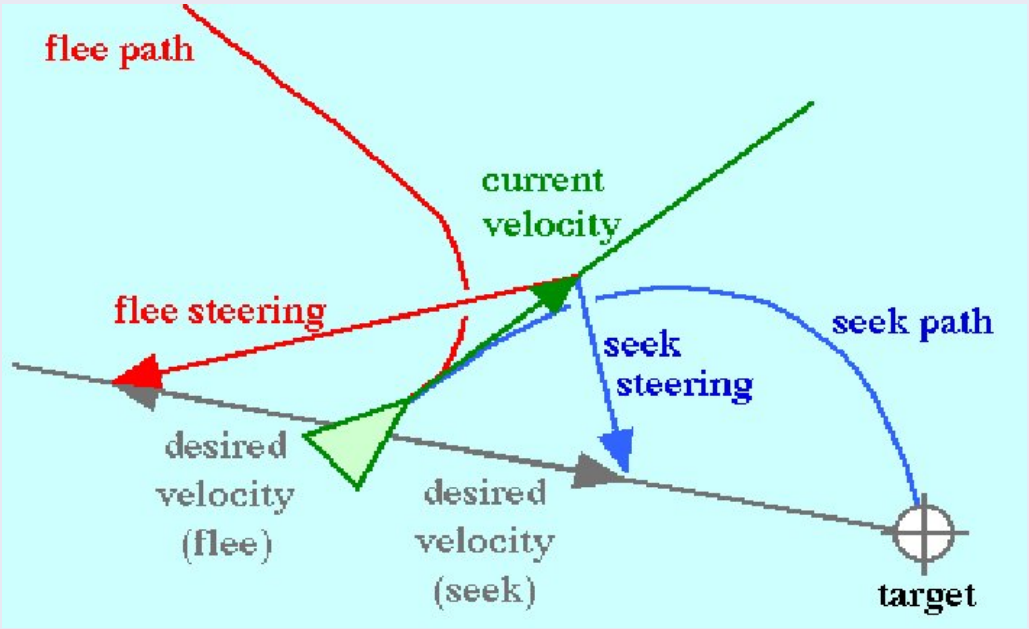
Mise à jour de la vélocité

$$\vec{\delta}(t+dt) = truncate(\vec{\delta}(t) + \frac{\vec{F_s}}{m}, \delta_{max})$$

Mise à jour de la position

$$P(t+dt) = P(t) + \vec{\delta}(t)$$

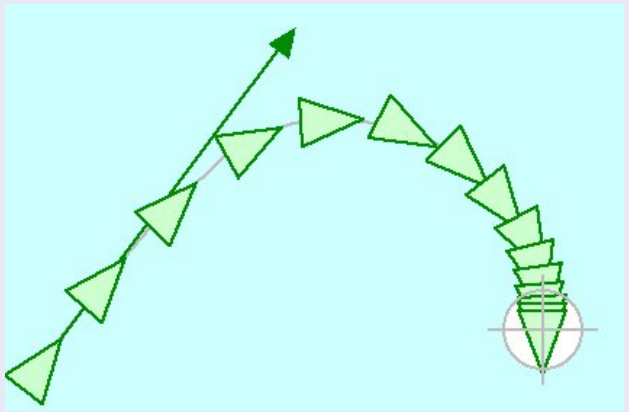
Modèle de steering
Passer par un point



$$\vec{F_s} = truncate(\vec{V_d} - \vec{V_c}, \vec{F_{max}})$$

$$V_d = \frac{\vec{AB}}{|\vec{AB}|}\delta_{max}$$

Modèle de steering
S'arréter sur un point



- Phénomènes d'oscillation
- Solution : réduction progressive de v_{max}

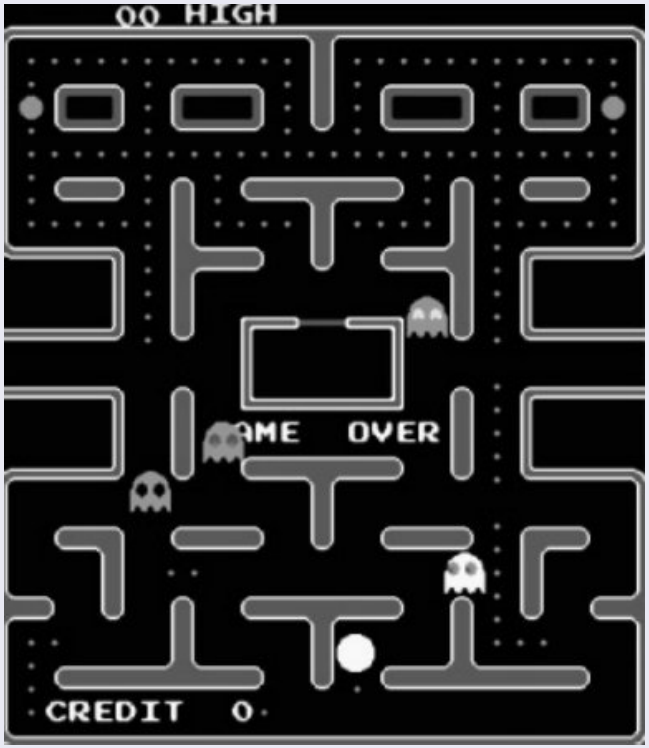
$$\vec{F_s} = truncate(\vec{V_d} - \vec{V_c}, \vec{F}_{max})$$

$$\vec{V_d} = \frac{\vec{AB}}{|\vec{AB}|}\delta_{max}\frac{d}{d_0}$$

$$d = min(d_0, |\vec{AB}|)$$

Modèle de steering
Modèle UML

Path-finding

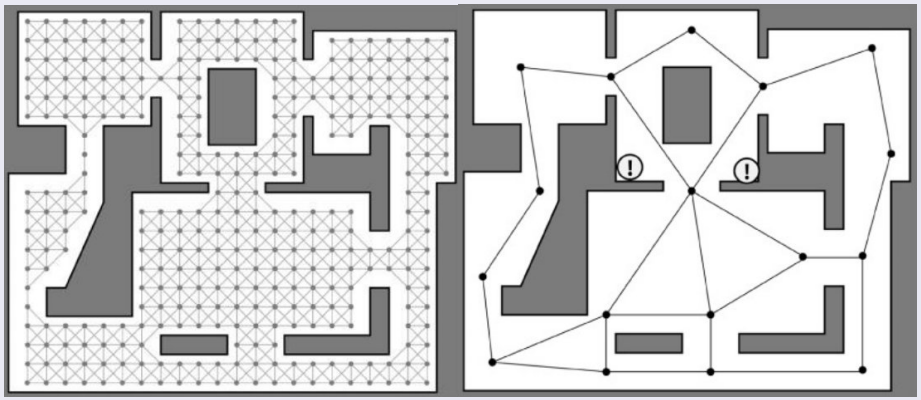


- Deux points P_0 et P_1
- Recherche d'un chemin entre P_0 et P_1 minimisant un critère
- Problème récurrent en RV, jeux, ...

Path-finding

Discrétisation de l'espace

- Echantillonnage régulier
- Echantillonnage irrégulier



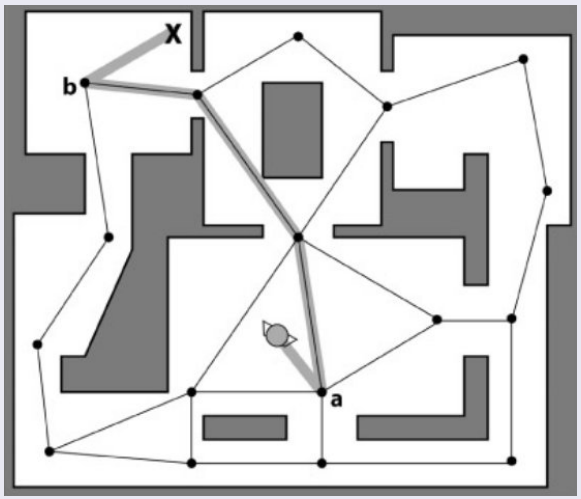
Connexions entre points

- X : ensemble des points
- Γ : connexions entre points
- $v(x, y)$: coût entre x et y
- $G = (X, \Gamma, v)$: graphe de navigation

Path-finding

Recherche d'un chemin

recherche d'un chemin de coût minimal dans un graphe



Algorithmes

Dijkstra, A^*

Path-finding

```
def A_STAR(G,s,r):
  in  : G = (V,E,c)
  out : pred
  for v in V :
    g(v) = inf
    pred(v) = None

  g(s) = 0
  S = {s}
  while not empty(S) :
    v : sommet de S minimisant g(v) + h(v,r)
    if v == r : return pred
    for u in voisins(G,v):
      if g(v) + c(u,v) < g(u):
        g(u) = g(v) + c(u,v)
        pred(u) = v
        S.add(u)
```

Automates

- Spécification des comportements
- Comportement : suite d'actions élémentaires
- Contextualité des comportements
- Utilisation de modèles formels : automates

Automates

Entité virtuelle

Objet qui se distingue de son environnement

Etat

- Position, orientation
- masse, aspect
- vitesse, accélération

Actions

- avancer, tourner
- passer par un point, atteindre un point
- suivre un chemin
- se montrer, se cacher
- dire un texte, changer de couleur,
- ...

Automates

Comportement d'une entité

Partie de son activité qui se manifeste à un observateur

Ensemble de ses actions/réactions dans une situation donnée.

Expression du omportement d'une entité

- Règles de production
- **SI** situation **ALORS** action
- situation : état , expression booléenne sur un contexte, ...
- action : transition vers un état, modification d'un contexte
- contexte : ensemble de variables caractérisant un objet

Automates

Exemple

- Par défaut Ttb se promène.
- si TTb voit un papillon il se dirige vers lui.
- Quand le papillon s'éloigne de lui Ttb recommence à se promener
- Si Ttb voit une souris il s'éloigne d'elle, effrayé.
- Quand la souris n'est plus visible il recommence à vagabonder.

Automates

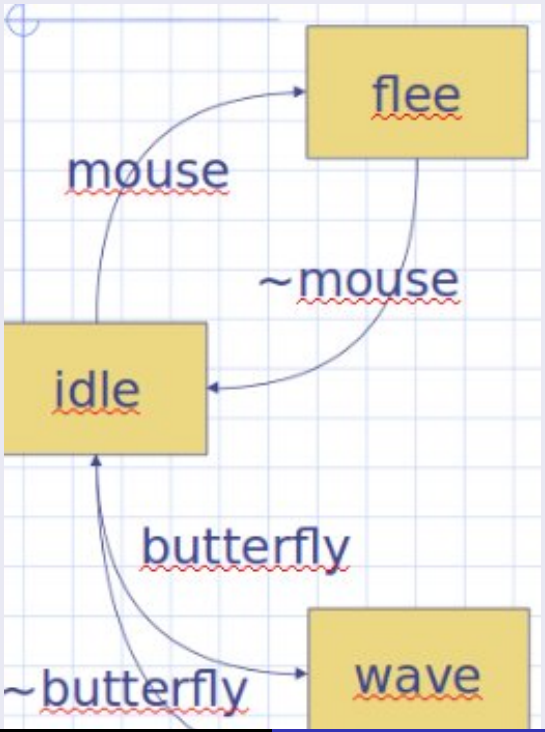
Etats

3 états : idle , flee , wave

Evénements

4 événements : mouse, notMouse, Butterfly, notButterfly

Automate



Description de comportement

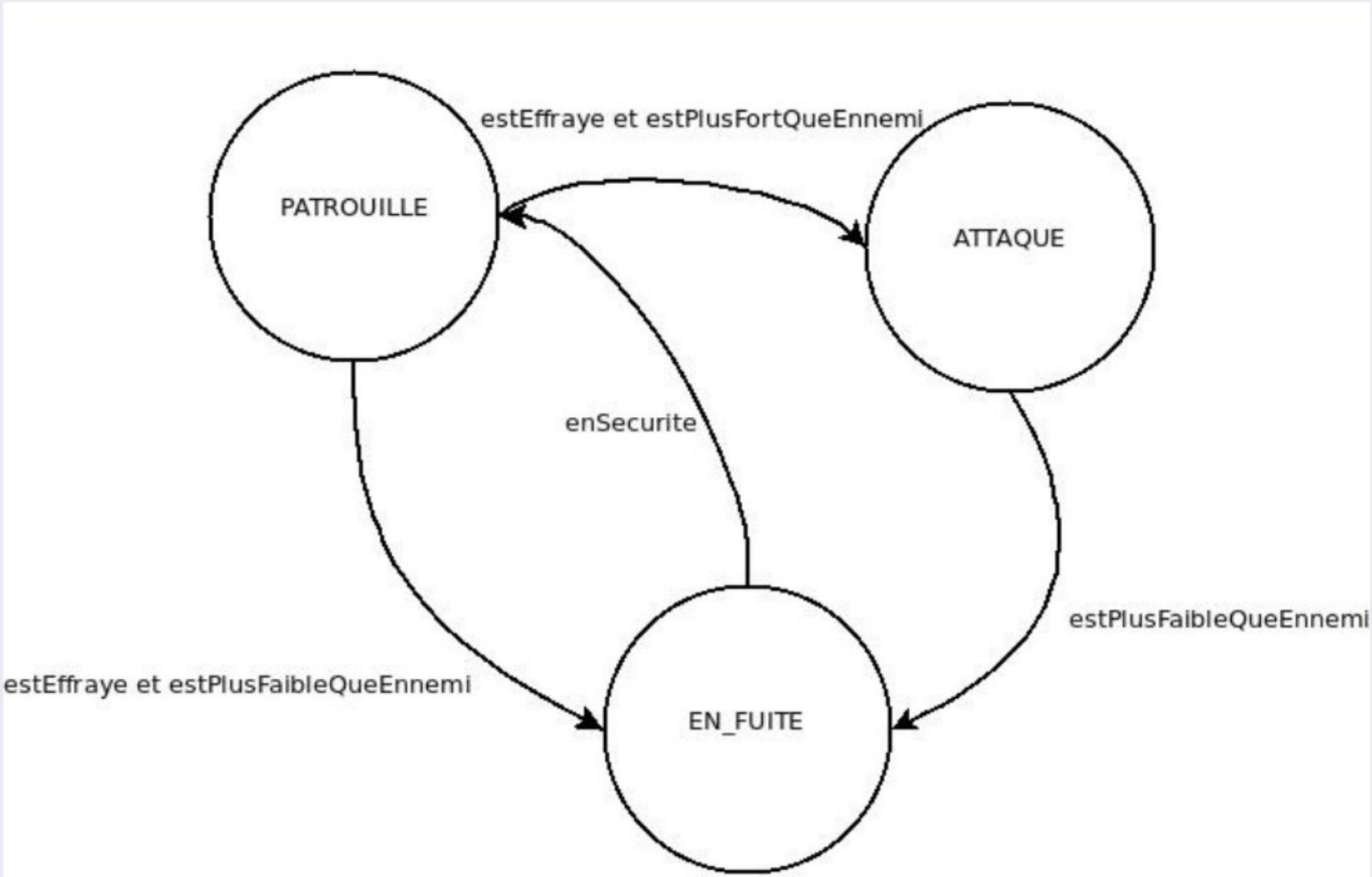
```
SI etat=FUITE et estPlusFaibleQueEnnemi()
  ALORS etat <- EN_FUITE

SI etat=PATROUILLE et estEffraye()
                   et estPlusFortQueEnnemi()
  ALORS etat <- ATTAQUE

SI etat=PATROUILLE et estEffraye()
                   et estMoinsFortQueEnnemi()
  ALORS etat <- EN_FUITE

SI etat=EN_FUITE et enSecurite()
  ALORS etat <- PATROUILLE
```

Automates


PATROUILLE
ATTAQUE
EN_FUITE
estEffraye et estPlusFortQueEnnemi
enSecurite
estPlusFaibleQueEnnemi
estEffraye et estPlusFaibleQueEnnemi

Automates

```
class FSM:
  def Execute(Acteur act):
    faire evoluer l etat de l automate
  def ChangeState(State newState):
    changer l etat de l automate

class Etat :
  def Enter(FSM fsm):
    code exécuté lorsqu'il devient l état courant
  def Execute(FSM fsm):
    code exécuté lorsqu'il est l état courant
  def Exit(Fsm fsm):
    code exécuté lorsqu'il arrête d'être état courant
```

Automates

```
class FSM:
  def __init__(self,acteur,e0):
    self.acteur = acteur
    self.currentState = e0
    e0.Enter(self)
    self.previousState = None

  def Execute(self):
    self.currentState.Execute(self)

  ...
```

Automates

```
def ChangeState(self,newState):
  self.previousState = self.currentState
  self.currentState.Exit(self)
  self.currentState = newState
  self.currentState.Enter(self)

def RevertToPreviousState(self):
  self.ChangeState(self.previousState)
```

Automates

```
class State :

  def Enter(self,fsm):
    pass

  def Execute(self,fsm):
    pass

  def Exit(self,fsm):
    pass
```

Automates

```
class Patrouille(State) :

  def __init__(self):
    State.__init__(self)

  def Execute(self,fsm):
    acteur = fms.acteur
    if          acteur.estEffraye() \
          and acteur.estPlusFortQueEnnemi() :
           fsm.ChangeState(Attaque())
    elif        acteur.estEffraye() \
           and acteur.estMoinsFortQueEnnemi():
           fsm.ChangeState(Fuite())
    else:
      acteur.tourner(30*signeAleatoire())
```

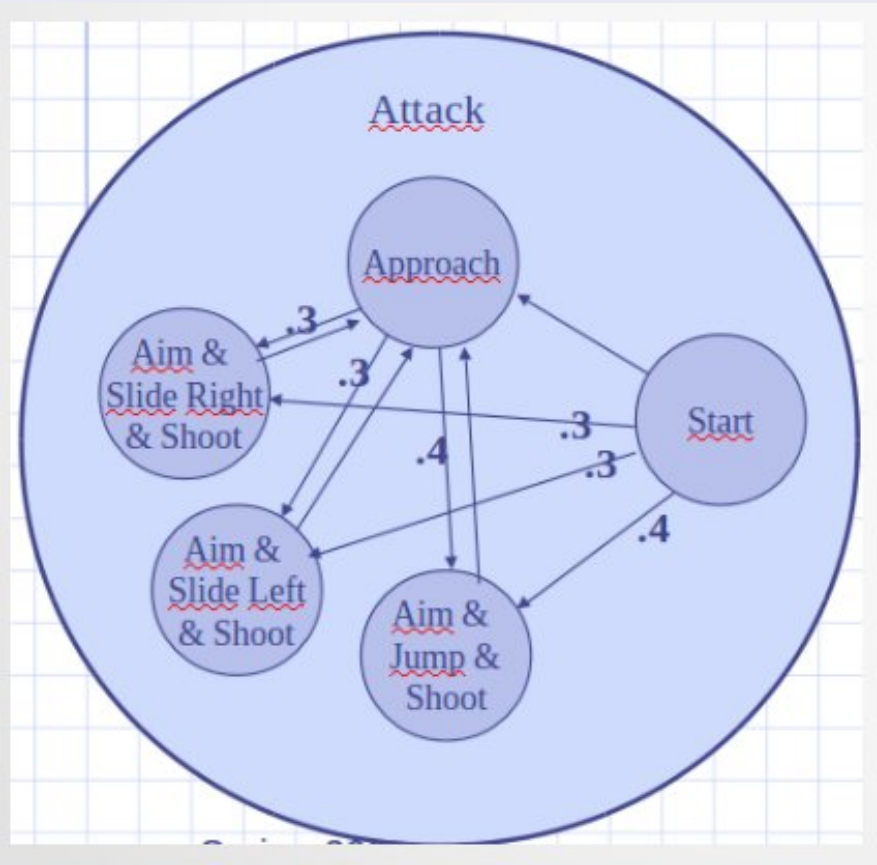
Automates

```
def Enter(self,fsm):
  acteur = fsm.acteur
  if acteur != None :
    acteur.setVitesse(2.0)

def Exit(self,fsm):
  pass
```

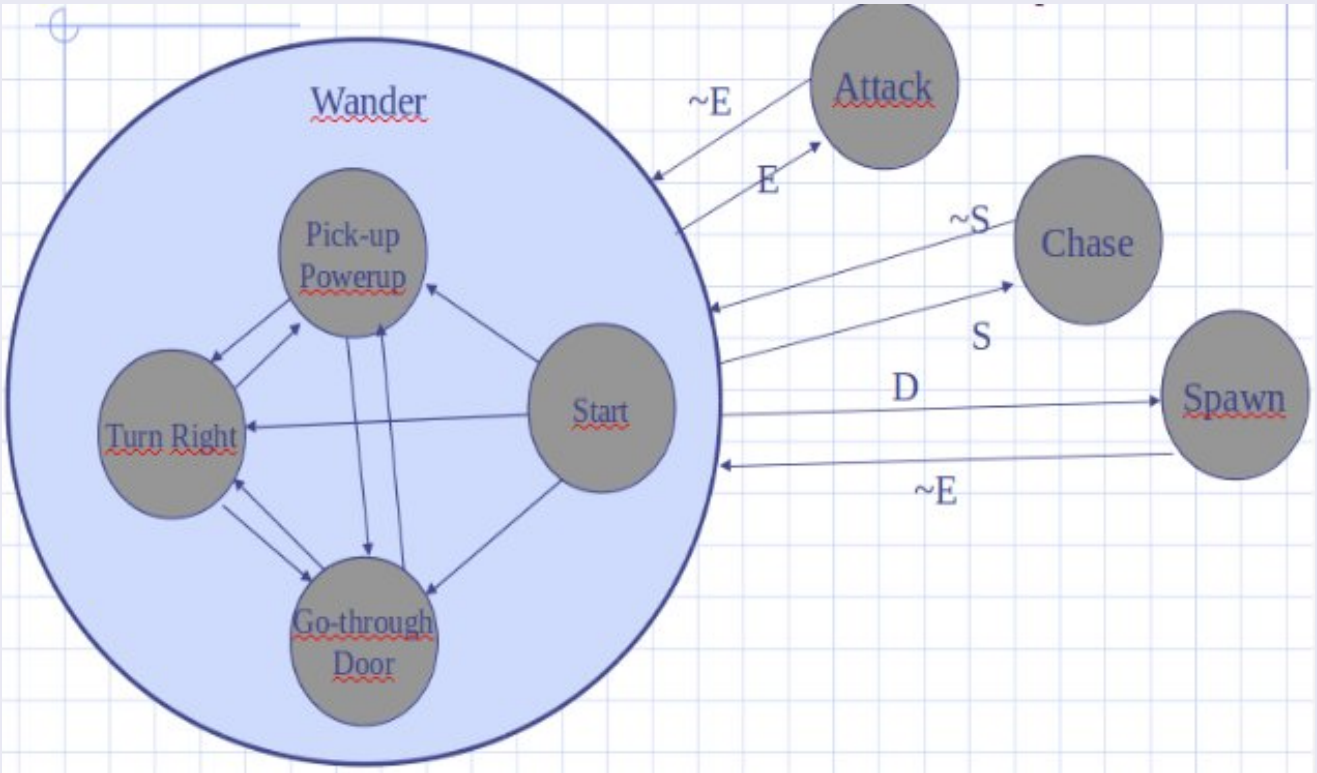
Automates

Automate Markovien

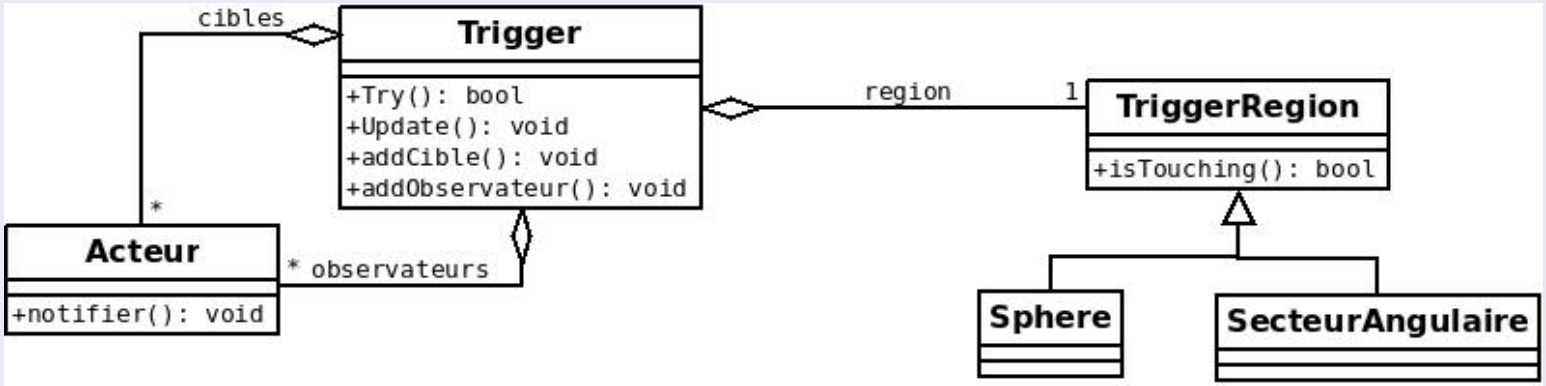


Automates

Automate hiérarchique



Capteurs virtuels



- Trigger : mise en oeuvre d'une Condition
- Région support

Capteurs virtuels

Sphère

Paramètres

- C : centre
- r : rayon

Critère

$$|\vec{CP}| < r$$

Capteurs virtuels

Secteur angulaire

Paramètres

- C : centre
- $\vec{D}$: direction
- h : horizon
- α : angle d'ouverture

Critère

$$|\vec{CP}| < h \ , \ \frac{dot(\vec{CP},\vec{D})}{|\vec{CP}|} > \cos(\frac{\alpha}{2})$$