esprit Se former autrement	**Atelier d'initiation à l'Intelligence Artificielle**
Matière : Projet 1A Classe(s) : 1A	Unité pédagogique : Algorithmique & Programmation Année universitaire : 2018-2019

Introduction

L'intelligence artificielle (IA) est la science dont le but est de créer ou simuler, chez les robots ou les logiciels, une intelligence et un comportement comparables à ceux de l'homme (penser, raisonner, voir, interagir, apprendre...) Elle cherche à construire des systèmes de plus en plus autonomes ainsi que des algorithmes capables de résoudre des problèmes complexes. Ainsi, une machine semble agir comme si elle était intelligente.

Il existe plusieurs domaines d'application de l'IA dont on cite la robotique, le diagnostic médical la reconnaissance de visage, parole, écriture et les jeux.

Dans le cadre des jeux vidéo, l'IA est utilisée pour contrôler les NPC (non player character) qui sont les ennemis et les entités secondaires. Ceci en leur permettant de se comporter d'une manière intelligente face aux différents scenarios possibles dans le jeu.

Dans cet atelier, nous allons nous contenter d'une initiation à l'utilisation de l'IA dans les jeux en introduisant le concept des machines à états finis.

1. Mise en contexte

Une machine ou automate à états finis (MEF) est un outil de modélisation formelle qui permet de programmer un mécanisme susceptible d'être dans un nombre fini d'états, mais étant un moment donné dans un seul état à la fois ; l'état dans lequel il se trouve alors est appelé l'«état courant». Le passage d'un état à un autre est activé par un événement ou une condition ; ce passage est appelé une « transition ».

Ainsi, une MEF est construite à partir des trois éléments de base suivants :

- **Des états** (de nombre est fini) **:** Un état est la description de la configuration d'un système en attente d'exécuter une transition.
- **Des transitions :** Une transition est un passage instantané d'un état courant vers un état suivant suite au déclenchement d'un événement.
- **Des évènements :** Ce sont les conditions permettant le passage d'un état à un autre.

Une MEF peut être représentée graphiquement par un **Diagramme d'états-transitions**. C'est un graphe comportant des états, matérialisés par des cercles ou des rectangles aux coins arrondis, et des transitions, matérialisées par des arcs orientés liant les états entre eux.

Exemples de mécanismes modélisables par une MEF : un compilateur, une machine à laver, le comportement dynamique d'une entité logicielle ...

Dans notre cas, il s'agit de la ou des valeurs d'une ou plusieurs variables (champs d'une structure). A un instant t précis, l'entité se trouve à un état précis.

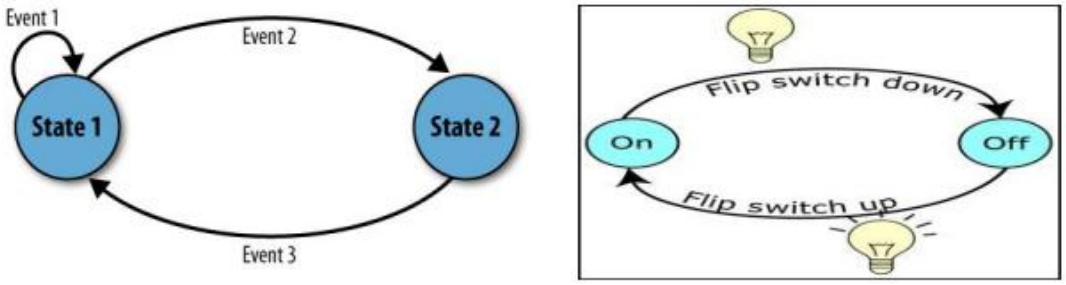


Figure 1. Exemple d'une MEF simple

2. Étude de cas 1 – Ennemi automatisé :

Le but de cet atelier est d'automatiser le comportement de l'ennemi. L'état de l'ennemi sera représenté à travers un champ qui devra être ajouté à la structure de l'ennemi. Cet état peut changer en fonction de la distance qui le sépare du héro. Les changements d'états de l'ennemi sont déclenchés par des évènements selon un certain scénario.

Le travail sera réalisé sur un projet initial **Atelier_IA_init.tar.gz** qui devra être enrichi au fur et à mesure en suivant les étapes de modélisation par MEF ci-après :

1) Élaborer le diagramme d'états – transitions de l'entité
2) Déduire sa table de vérité en précisant pour chaque transition : l'état courant, l'état suivant et les évènements de déclenchement
3) Traduire la table de vérité dans la mise à jour de l'ennemi.

a. scénario 1

On se propose le scénario suivant :

"L'ennemi est en attente quand soudain le héro se trouve dans son champs de vision. A ce moment là, l'ennemi court à la poursuite du héro. Lorsqu'il devient assez proche, l'ennemi commence ces attaques acharnées. Une fois qu'il a touché le héro (que le héro soit mort ou qu'il a pris la fuite), l'ennemi revient à son état d'attente jusqu'a ce que le héro soit de nouveau dans son champ de vision".

D’après ce scénario, le critère du choix des événements et en relation directe avec la distance ***d*** séparant l’ennemi du héro. Pour ceci nous allons fixer deux seuils de distance ***s1*** et ***s2*** :

- ***s1*** : seuil de distance à partir duquel le personnage est dans le champ de vision de l’ennemi, ce qui permettra à ce dernier de faire la poursuite.
- ***s2*** tel que ***s2*** < ***s1*** : ***s2*** est un seuil de distance plus proche du héro. A partir de ***s2***, l'ennemi va attaquer le héro.

La figure suivante présente quelques illustrations de l'état de l'ennemi en fonction des sa distance par rapport au héro. Dans cet exemple, ***s1*** = 600 et ***s2*** = 100.

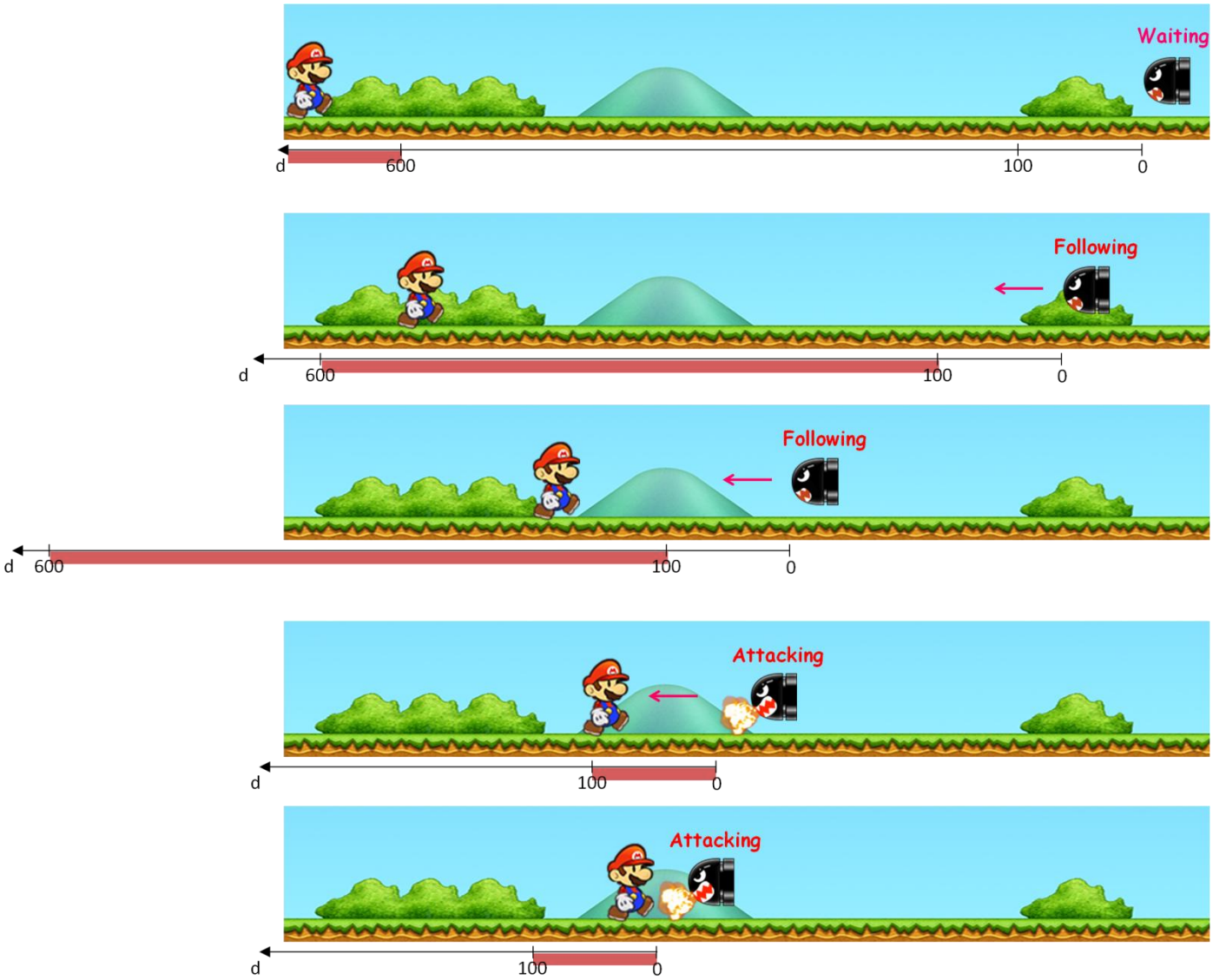


Figure 2. Illustrations du scénario1

b. Traitement des transitions :

Les évènements liés au déplacement du héro par l'utilisateur (et éventuellement ceux liés au déplacement de l'ennemi même) vont par la suite agir sur la distance ***d*** qui sépare le héro de l'ennemi.

- Initialement, l'ennemi est en état de ***Waiting*** et restera en cet état tant que d est supérieure au seuil s1 (600 dans scénario1).

  ⇛ Transition ***t1***: l'ennemi reste en état de ***Waiting***

- si la distance d est comprise entre s2 et s1, le héro se trouve dans le champs de vision de l'ennemi, ce qui engendrera la transition t2 :

  ⇛ Transition ***t2*** : Transition de l'état ***Waiting*** à l'état ***Following***.

- Lorsque l'ennemi est en état de Following et la distance qui le sépare au personnage est comprise entre d1 et d2. Dans ce cas l'ennemi continue encore sa poursuite au personnage du jeu.

  ⇛ Transition ***t3***: l'ennemi reste en état de ***Following***

- Lorsque l'ennemi est en état de Following et il est trops prés du personnage de jeu (distance < d2). Dans ce cas il peut bien attaquer donc transition à l'état d'attaque:

  ⇛ Transition ***t4*** : Transition de l'état ***Following*** à l'état ***Attacking***

- Lorsque l'ennemi est en état d'attacking et la distance qui le sépare au personnage est toujours infèrieur à d2. Dans ce cas l'ennemi continue encore l'attaque du personnage du jeu.

  ⇛ Transition ***t5*** : l'ennemi reste en état ***Attacking***

- Lorsque l'ennemi est en état d'attacking et à partir du moment où la distance qui le sépare au personnage devient nulle (c'est-à-dire, à partir de la collision entre l'ennemi et le héro), l'ennemi revient à son état initial qui est l'état Waiting.

  ⇛ Transition ***t6*** : Transition de l'état ***Attacking*** à l'état ***Waiting***

Algorithme de traitement des transitions

SELON etat_ennemi **FAIRE**

CAS Waiting :

// transition t1 : Waiting → Waiting

SI $d > s1$ **ALORS** etat_ennemi = Waiting **FINSI**

// l'etat reste inchangé → inutile d'implémenter ce cas

// transition t2 : Waiting → Following

SI $s2 < d \leq s1$ **ALORS** etat_ennemi = Following **FINSI**

CAS Following :

// transition t3 : Following → Following

SI $s2 < d \leq s1$ **ALORS** etat_ennemi = Following **FINSI**

// l'etat reste inchangé → inutile d'implémenter ce cas

// transition t4 : Following → Attacking

SI $0 < d \leq s2$ **ALORS** etat_ennemi = Attacking **FINSI**

CAS Attacking :

// transition t5 : Attacking → Attacking

SI $0 < d \leq s2$ **ALORS** etat_ennemi = Attacking **FINSI**

// l'etat reste inchangé → inutile d'implémenter ce cas

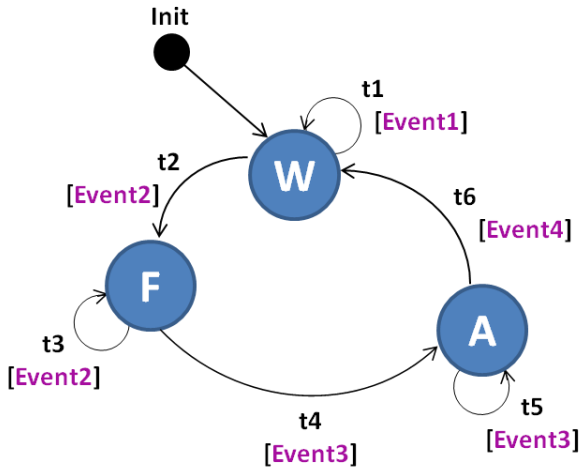
// transition t6 : Attacking → Waiting

SI $d \leq 0$ **ALORS** etat_ennemi = Waiting **FINSI**

FIN SELON

c. Diagramme d'états – transitions :

Les changements d'états de l'ennemi par le scénario 1 sont représentées par le diagramme d'états – transitions suivant :



Event1 : d > 600
Event2 : 100 < d ≤ 600
Event3 : 0 < d ≤ 100
Event4 : d ≤ 0

Figure 3. MEF du scénario1

d. Table de vérité :

La table de vérité permet de détailler pour chaque transition, l'état courant de l'entité traitée et les évènements responsables de son déclenchement, ainsi que l'état suivant de l'entité.

Table 1. Table de vérité du scénario1

Transition	État courant	Event1	Event2	Event3	Event4	État suivant	
t1	W	1	0	0	0	W	← *Aucun changement d'état*
t2	W	0	1	0	0	F	
t3	F	0	1	0	0	F	← *Aucun changement d'état*
t4	F	0	0	1	0	A	
t5	A	0	0	1	0	A	← *Aucun changement d'état*
t6	A	0	0	0	1	W	

d. Implémentation :

Pour implémenter une MEF de l'entité Ennemi dont on veut gérer le comportement, il faut :

- Ajouter un attribut State à la structure Ennemi qui prend sa valeur à partir d'une liste d'énumérations {Waiting, Following, Attacking}. Initialiser State à l'état Waiting.

- Implémenter la fonction ***updateEnnemi ( )*** qui permet de mettre à jour un ennemi en fonction des évènements (de déplacement du héro et de l'ennemi) et en fonction de l'état courant de l'ennemi. La mise à jour concerne aussi bien l'animation, le déplacement ainsi que le nouvel état de l'ennemi. L'animation devrait également être modifiée selon l'état de l'ennemi (attaque / non attaque) pour afficher les images correspondantes. La mise à jour de l'état se fait par l'implémentation de la MEF dans la fonction ***updateEnnemiState ( )*** à compléter.

ennemi.h

```
typedef enum STATE STATE;
enum STATE {WAITING, FOLLOWING,ATTACKING};

struct ennemi
{
        SDL_Surface * image;
        SDL_Rect positionAnimation [SPRITE_ENNEMI_NbL][SPRITE_ENNEMI_NbCol];
        FRAME Frame;
        SDL_Rect positionAbsolue;
        int Direction;
        STATE State;
};
typedef struct ennemi Ennemi;
```

ennemi.c.

```
void update_ennemi(Ennemi* E, SDL_Rect posHero)
{
        int distEH = E->positionAbsolue.x - (posHero.x + Hero_WIDTH);
        printf("distEH = %d\t E->State = %d\n", distEH,E->State);

        switch(E->State)
        {
                case WAITING :
                {
                        animateEnnemi(E);
                        break;
                }

                case FOLLOWING :
                {
                        animateEnnemi(E);
                        moveEnnemi(E,posHero);
                        break;
                }

                case ATTACKING :
                {
                        // Attaque en se déplaçant vers l'ennemi
                        animateEnnemi(E);
                        moveEnnemi(E,posHero);
                        break;
                }
        }

        updateEnnemiState(E, distEH);
}
```

Travail à faire :

Dans le projet **Atelier_IA_init.tar.gz**, Implémenter la fonction ***updateEnnemiState ( )*** et enrichir la mise à jour de l'ennemi en fonction de son état.

2. Étude de cas 2 :

a. scénario2

"L'ennemi est en attente quand soudain le héro se trouve dans son champs de vision. A ce moment là, l'ennemi commence à faire des allers/retours (éventuellement par des déplacements aléatoires). Lorsque l'ennemi et le héro deviennent plus proches, l'ennemi court à la poursuite du héro. Lorsqu'il devient encore plus proche, l'ennemi commence ces attaques acharnées tant que le héro n'est pas en train d'attaquer à son tour. Par contre, l'ennemi prend la fuite dès que le héro l'attaque. Dans le cas où l'ennemi a commencé ses attaques, une fois qu'il a touché le héro, l'ennemi revient à l'état de déplacement par des allers/retours et si le héro s'éloigne d'avantage de lui, il revient à l'état d'attente, jusqu'a ce que le héro soit de nouveau dans son champ de vision".

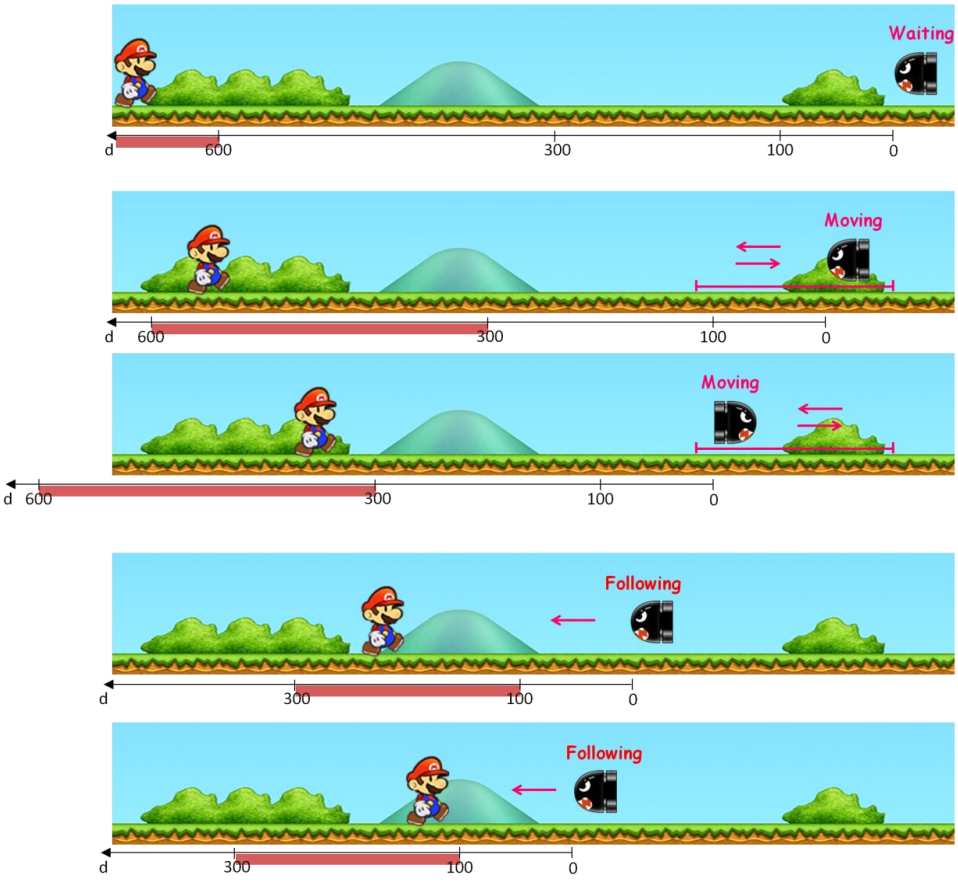


Figure 3. Illustrations du scénario2 : états Waiting et Following

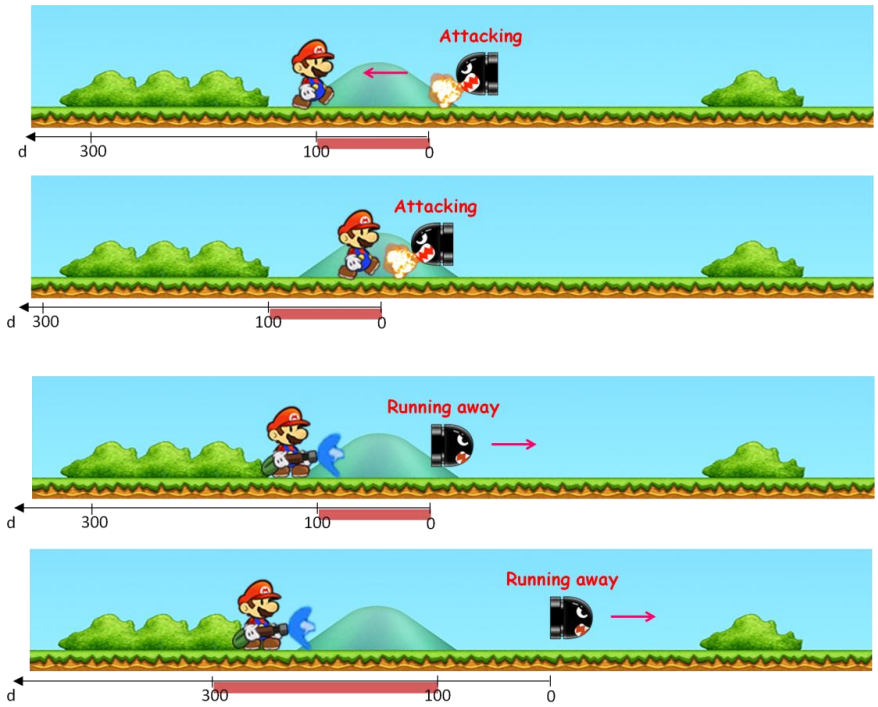
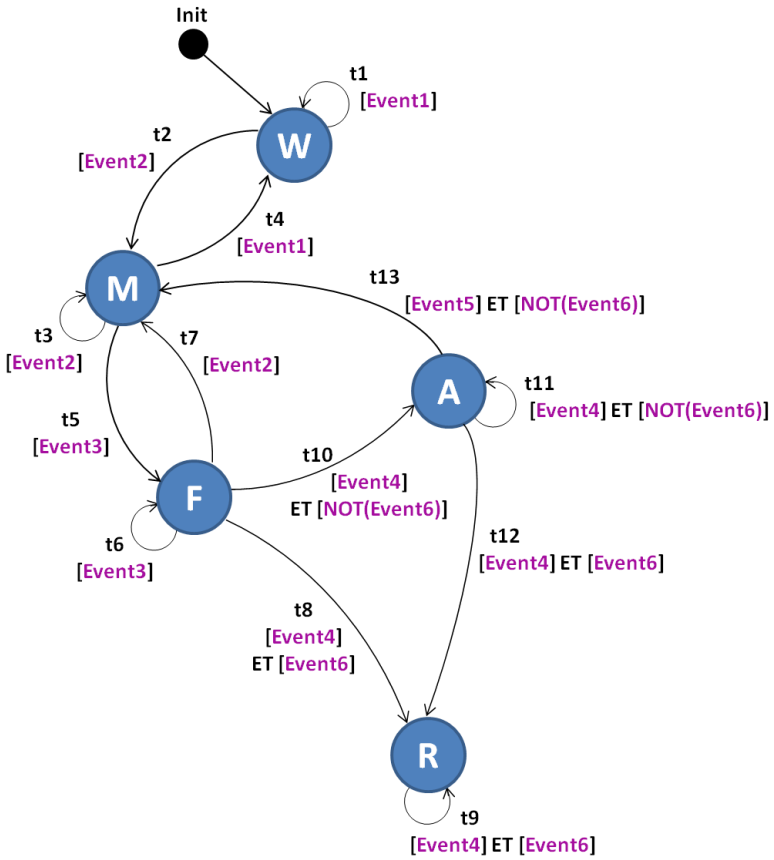


Figure 4. Illustrations du scénario2 : états Attacking et Run away

b. Diagramme d'états – transitions :

Les changements d'états de l'ennemi par le scénario 1 sont modélisés comme suit :



Event1 : d > 600

Event2 : 300 < d ≤ 600

Event3 : 100 < d ≤ 300

Event4 : 0 < d ≤ 100

Event5 : d ≤ 0

Event6 : hero_attack = 1

Figure 5. MEF du scénario2

c. Table de vérité :

La table de vérité du scénario 2 est déduite du diagramme d'états – transitions précédent comme suit :

Table 2. Table de vérité du scénario2

Transition	État courant	Event1	Event2	Event3	Event4	Event5	Event6	État suivant
t1	W	1	0	0	0	0	*	W
t2	W	0	1	0	0	0	*	M
t3	M	0	1	0	0	0	*	M
t4	M	1	0	0	0	0	*	W
t5	M	0	0	1	0	0	*	F
t6	F	0	0	1	0	0	*	F
t7	F	0	1	0	0	0	*	M
t8	F	0	0	0	1	0	1	R
t9	R	0	0	0	1	0	1	R
t10	F	0	0	0	1	0	0	A
t11	A	0	0	0	1	0	0	A
t12	A	0	0	0	1	0	1	R
t13	A	*	*	*	*	1	0	M

d. Implémentation (travail à faire):

Automatiser le comportement des ennemis de votre jeu avec un scénario au moins aussi riche que celui modélisé par diagramme d'états – transitions du scénario2.