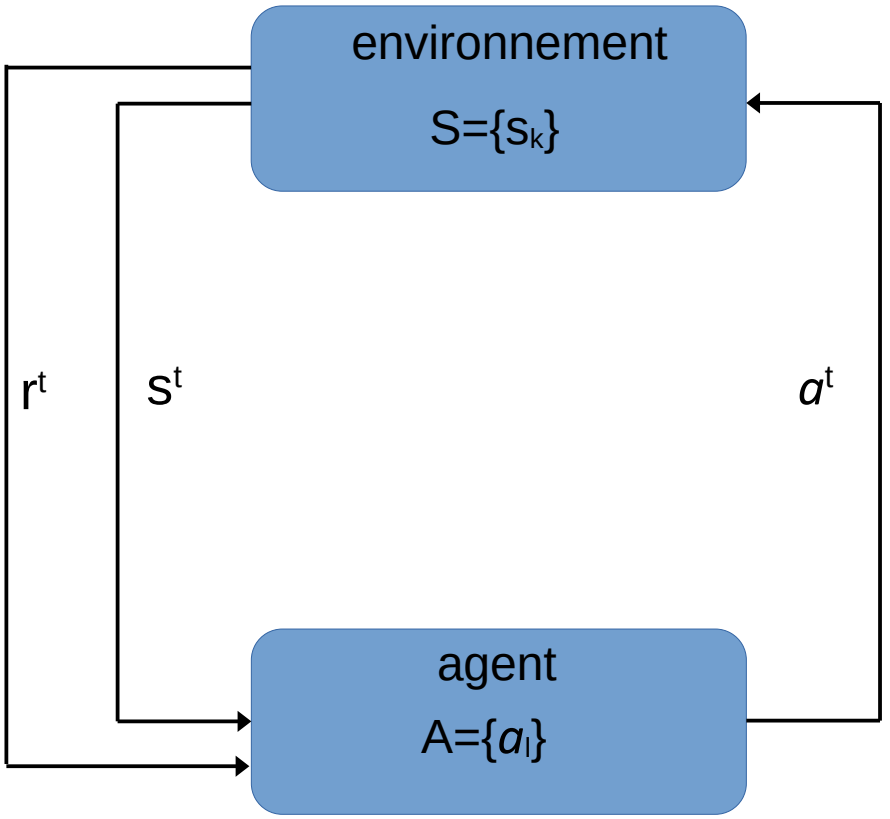
Cours-TD d'introduction à l'Intelligence Artificielle

Partie VII

L'apprentissage par renforcement

Simon Gay

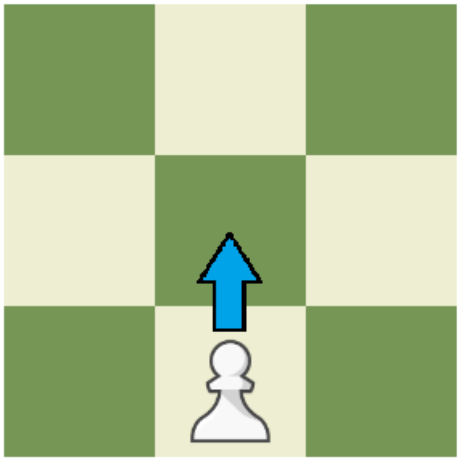
Introduction à l'Intelligence Artificielle

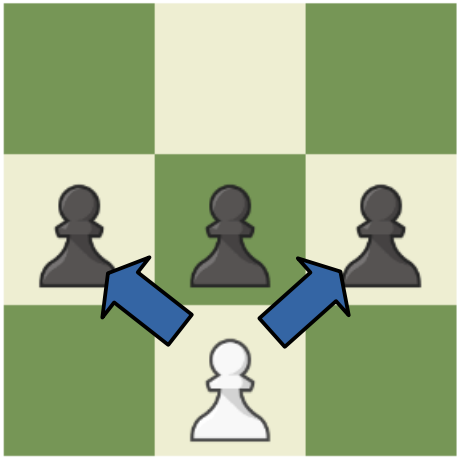


Passons à la pratique !

Introduction à l’Intelligence Artificielle

- **Sujet d’étude**
  - Jeu de l’Hexapion
    - Un damier de 3x3 cases
    - 3 pions VS 3 pions
    - Mêmes déplacements que pour le jeu d’échec





Introduction à l’Intelligence Artificielle

- **Sujet d’étude**
  - Jeu de l’Hexapion
    - Deux façons de gagner :
      - En atteignant le côté adverse
      - En mangeant tous les pions adverses

Introduction à l'Intelligence Artificielle

- **Sujet d'étude**
  - Considérations techniques :

    - Le plateau est une matrice de 3x3 cases, n_i = 1 pour un pion blanc, -1 pour un pion noir et 0 pour une case vide

    - Nombre d'états théoriques : 3^9 = 19 683
      - En pratique, il y en a beaucoup moins (environ 200) : nombre de pions limités et contraintes sur les déplacements

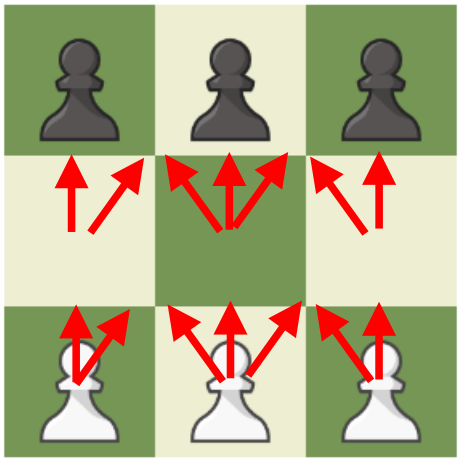
        → on utilisera un tableau dynamique pour stocker les états découverts

    - On définit un identifiant unique pour chaque état possible de la façon suivante :

$$id(s) = (n_8+1) \times 3^8 + (n_7+1) \times 3^7 + (n_6+1) \times 3^6 + \ldots + (n_0+1) \times 3^0$$

Introduction à l'Intelligence Artificielle

- **Sujet d'étude**
  - Considérations techniques :



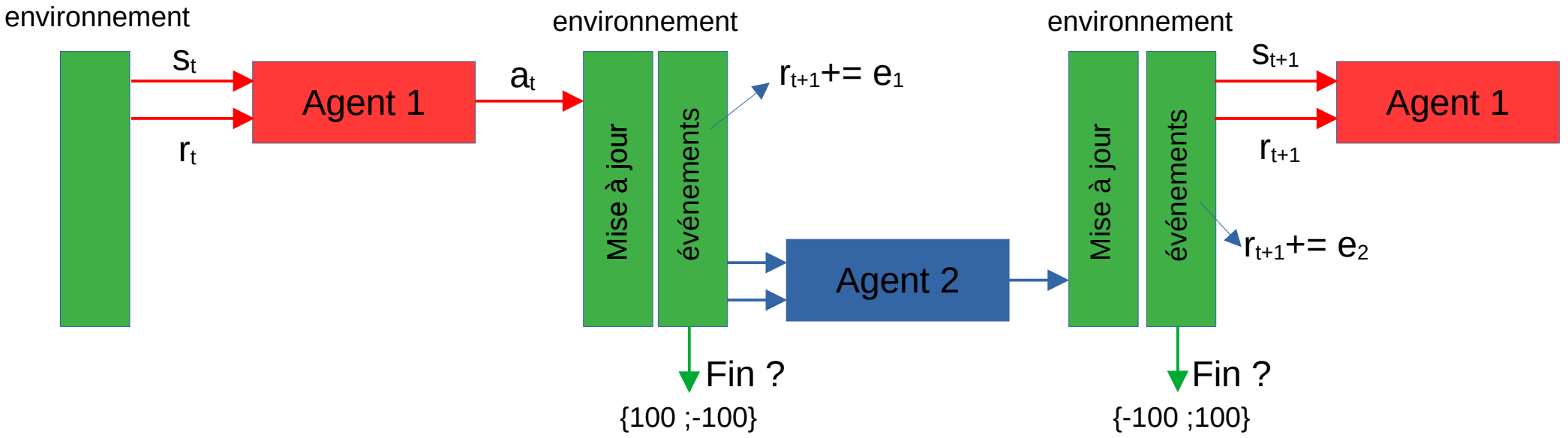
    - 14 actions possibles :
      - Seule une partie est disponible à un instant donné

    - Nous considérerons les récompenses suivantes :

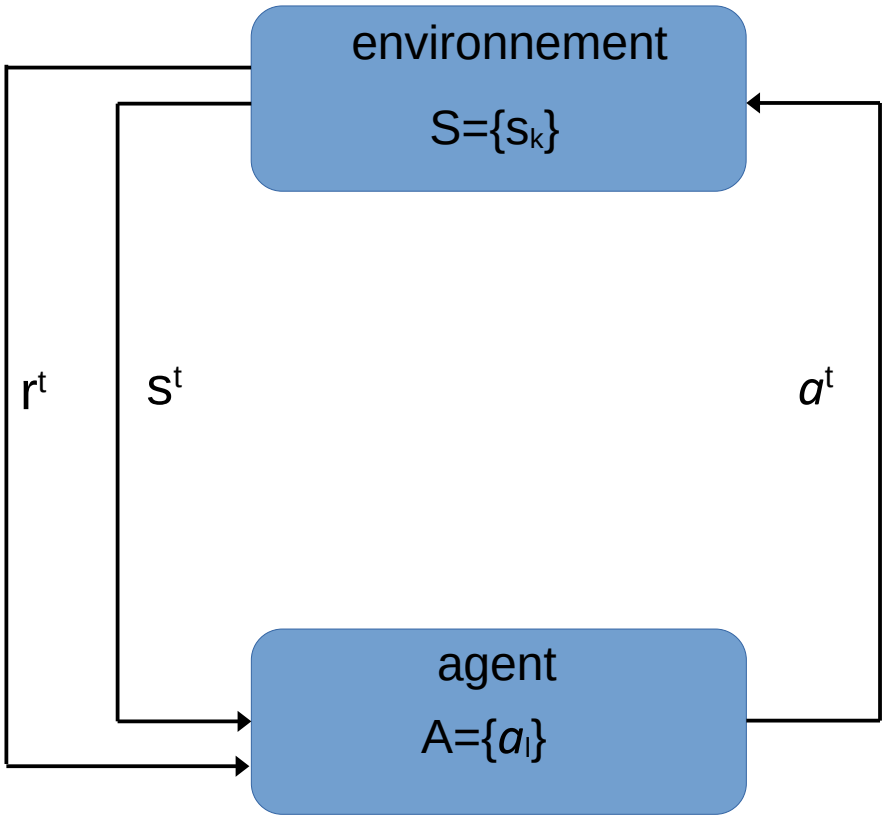
      +10 si le joueur mange un pion
      -10 si le joueur adverse mange un pion
      +100 pour une victoire
      -100 pour une défaite
      +10 pour un match nul

Introduction à l'Intelligence Artificielle

- **Sujet d'étude**
  - Considérations techniques : présence d'un adversaire → environnement dit *hostile*
    - Les joueurs jouent à tour de rôle : un joueur effectue une action, mais reçoit l'état suivant et la récompense après l'action de l'autre joueur
    - Timeline (côté joueur 1) :



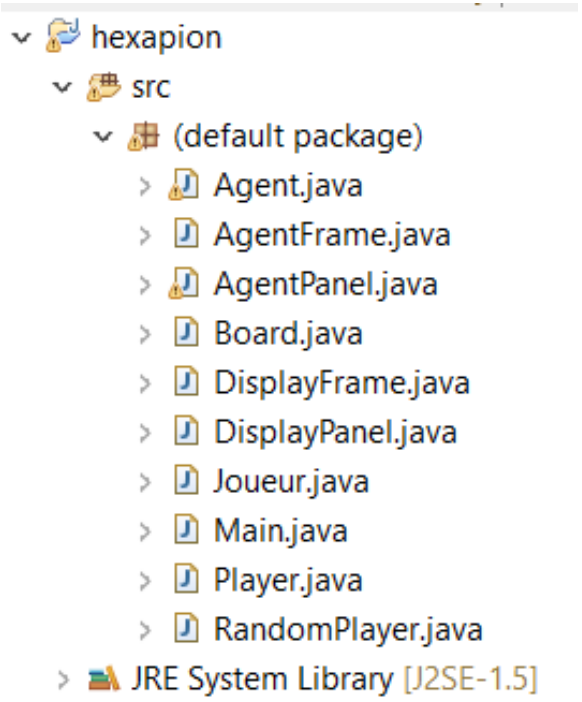
Introduction à l'Intelligence Artificielle



Commençons à coder !

Introduction à l'Intelligence Artificielle

- **L'environnement**
  - Ouvrez Eclipse
  - Créez un nouveau projet Java 'hexapion'
    - Clic droit dans package explorer → new → Java project
  - Télécharger l'archive hexapion.zip sur le github, puis importez les 10 classes java dans votre nouveau projet.



Introduction à l’Intelligence Artificielle

- **L’environnement**
  - Étudions les classes à notre disposition :

    - Main : classe principale. C’est elle qui orchestre la partie, récupère les actions des agents, met à jour le plateau de jeu et attribue les récompenses

      → TODO : la gestion des récompenses

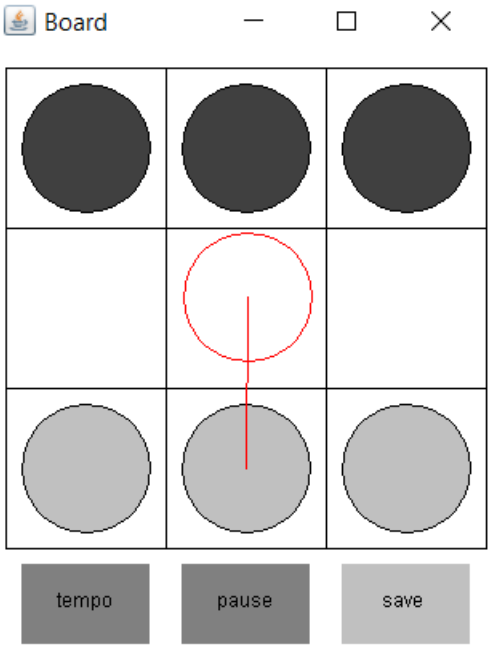
    - Board : classe contenant le plateau de jeu et les fonctions pour convertir les actions et leurs identifiants.

      → rien à faire !

Introduction à l'Intelligence Artificielle

- **L'environnement**
  - Étudions les classes à notre disposition :

    - DisplayFrame et DisplayPanel : classes pour l'afficheur principal. Gère les boutons de fonction et l'interface utilisateur



Introduction à l’Intelligence Artificielle

- **L’environnement**
  - Étudions les classes à notre disposition :
    - Player : classe générique d’un joueur. Définit les fonctions principales :
      - initialize : pour préparer le joueur à une nouvelle partie
      - update : récupération de l’état et de la récompense, retourne l’action
      - learn : apprentissage d’une récompense issue d’un état terminal (nous verrons plus tard à quoi elle sert)
      - save : fonction utilitaire pour sauvegarder les paramètres d’un agent.

Introduction à l'Intelligence Artificielle

- **L'environnement**
  - Étudions les classes à notre disposition :
    - Les trois classes suivantes héritent de la classe Player :
      - RandomPlayer est un joueur qui retourne une action au hasard
      - Joueur est une interface pour l'utilisateur (vous pourrez jouer contre votre propre IA)
      - Agent est la classe contenant l'algorithme d'apprentissage
        - Pour le moment, elle ne fait rien de plus que RandomPlayer
        - Elle dispose de son propre afficheur (AgentFrame et AgentPanel)

        → TODO : les algorithmes d'apprentissage

Introduction à l'Intelligence Artificielle

- **L'environnement**
  - Familiarisons-nous avec l'environnement :

    - La classe Main permet de choisir le type des deux joueurs.
      - Vous pouvez voir s'affronter deux joueurs aléatoires, ou jouer contre un joueur aléatoire

```
public Main(){

    board=new Board();  // initialize game board

    ////////////////////////////////////////////////
    // select two players

    player1=new RandomPlayer("AgentA", 1, this);
    //player1=new Joueur("AgentA", 1, this);

    player2=new RandomPlayer("AgentB", 2, this);
    //player2=new Joueur("AgentB", 2, this);

    ////////////////////////////////////////////////
```

Introduction à l'Intelligence Artificielle

- **L'environnement**
  - Gestion des récompenses :
    - On utilise les deux variables *reward1* et *reward2* pour accumuler les récompenses entre deux décisions.
    - Après lecture de la récompense, on réinitialise la variable
    - On intègre les récompenses liées aux évènements.

```
// perform action
board.movePlayer1(action);

display.repaint();
pause();

reward1=0;  // reset reward

// get reward from move (eat events)
if (board.event==1){
    reward1+= 10;    // add reward for player1
    reward2+=-10;    // add reward for player2
}
```

```
// perform action
board.movePlayer2(action);

display.repaint();
pause();

reward2=0;  // reset reward

// get reward from move (eat events)
if (board.event==1){
    reward1+=-10;
    reward2+= 10;
}
```

    - On n'oublie pas de réinitialiser les récompenses à la fin d'une partie (pour la suivante)

```
board.initialize();
player=1;

reward1=0;
reward2=0;

player1.initialize();
player2.initialize();
gameover=false;
```

Introduction à l'Intelligence Artificielle

- **L'agent**

  - Nous allons enregistrer les états découverts par l'agent
  - En Java, il existe une classe de vecteur dynamique très pratique : le ArrayList

  - On peut faire un ArrayList avec à peu près n'importe quoi :
    - Des valeurs (on utilisera les types **Integer**, **Float**, **Byte** au lieu de **int**, **float**, **byte**)
    - Des tableaux : **int[ ]**, **float[ ]**, **int[ ][ ]** …
    - Plus généralement, avec tout type d'objet (on peut faire un ArrayList d'*Agent*)

Introduction à l'Intelligence Artificielle

- **L'agent**

  - Déclaration : `public ArrayList<Integer> states;`
  - Initialisation : `states=new ArrayList<Integer>();`

  - Ajouter un élément : `states.add( 5 );`
  - Insérer un élément à une position donnée : `states.add( i, 5 );`

  - Récupérer un élément de la liste : `states.get(i);`
    - Dans le cas d'un ArrayList<int[ ]> : `states.get(i)[j]=5;`
  - Taille de la liste : `states.size();`
    - Équivalent au 'length' des tableaux
  - Index de la première occurrence : `states.indexOf(5);`
    - Retourne -1 si non présent
  - Effacer le contenu : `states.clear();`

Introduction à l'Intelligence Artificielle

- **L'agent : la Q-table**

  - Nous allons créer une Q-table dynamique qui va enregistrer :

    - Le numéro de l'état (pour éviter les doublons)
      → ArrayList de **Integer**

    - Les Q-values de chaque action possible dans cet état
      → ArrayList de **float[ ]**

    - Les numéros des actions possibles dans cet état
      → ArrayList de **int[ ]**

  - La liste des actions possibles est donnée par le paramètre *possible_actions* (de type ArrayList) de la fonction *update*

Introduction à l'Intelligence Artificielle

- **L'agent : la Q-table**

  - Déclaration :

```
public ArrayList<Integer> states;
public ArrayList<int[]> actions;
public ArrayList<float[]> Qtable;
```

  - Initialisation :

```
public Agent(String name, int id, Main m){
    super(name, id,m);

    states=new ArrayList<Integer>();
    actions=new ArrayList<int[]>();
    Qtable=new ArrayList<float[]>();
```

Introduction à l'Intelligence Artificielle

- **L'agent : la Q-table**

  - Remplissage de la Q-table par la fonction update :

```
public int update(int state, int reward, ArrayList<Integer> possible_actions){

    // if unknown state, add it to the list
    if (states.indexOf(state)==-1){

        states.add(state);

        // conversion ArrayList -> tableau
        int[] actList=new int[possible_actions.size()];
        for (int i=0;i<possible_actions.size();i++) actList[i]=possible_actions.get(i);

        // vecteur de float à 0
        float[] table=new float[possible_actions.size()];

        actions.add(actList);
        Qtable.add(table);
    }
```

Introduction à l'Intelligence Artificielle

- **L'agent : lecture de la Q-table**

  - On prépare deux variables pour stocker l'index de l'état actuel dans la table et de l'action choisie

```
public ArrayList<Integer> states;
public ArrayList<int[]> actions;
public ArrayList<float[]> Qtable;

public int stateIndex=-1;
public int actionIndex=-1;
```

  - Ces variables doivent être réinitialisées quand on débute une nouvelle partie

```
public void initialize(){
    stateIndex=-1;
    actionIndex=-1;
}
```

Introduction à l'Intelligence Artificielle

- **L'agent : lecture de la Q-table**

  - Dans update, on récupère l'index de l'état courant après la partie ajout des états

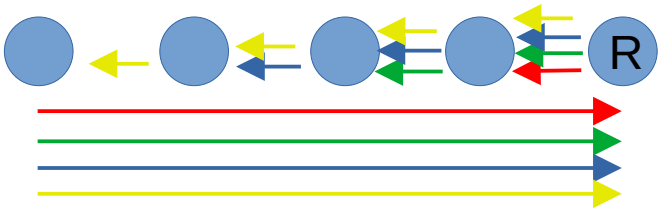
```
// if unknown state, add it to the list
if (states.indexOf(state)==-1){

    states.add(state);

    // conversion ArrayList -> tableau
    int[] actList=new int[possible_actions.size()];
    for (int i=0;i<possible_actions.size();i++) actList[i]=possible_actions.get(i);

    // vecteur de float à 0
    float[] table=new float[possible_actions.size()];

    actions.add(actList);
    Qtable.add(table);
}
```

```
// get new state index
stateIndex=states.indexOf(state);
```

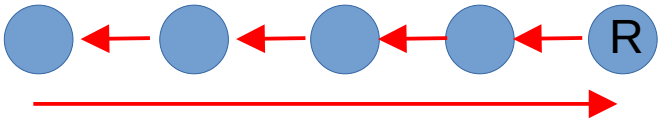
Introduction à l'Intelligence Artificielle

- **L'agent : Apprentissage des Q-values**

  - Nous allons optimiser l'apprentissage :
    - Dans l'exemple vu en cours, il faut passer plusieurs fois sur le chemin optimal pour propager les Q-values au travers des états



    - Comme les séquences d'action ont un début et une fin, on va pouvoir utiliser une astuce : on enregistre la séquence des états, actions et récompenses
    - Puis on backpropage les Q-values en appliquant la séquence en sens inverse



$\{ \{s_0,r_1,a_1\}, \{s_1,r_2,a_2\}, \{s_2,r_3,a_3\}, \{s_3,r_4,a_4\}, \{s_4,r_5,a_5\} \}$

    - Nous allons stocker les valeurs nécessaires

Introduction à l'Intelligence Artificielle

- **L'agent : Apprentissage des Q-values**

  - 3 ArrayList pour stocker les états, les récompenses et les actions
    - Déclaration :

```
public int stateIndex=-1;
public int actionIndex=-1;

public ArrayList<Integer> actionSequence;
public ArrayList<Integer> stateSequence;
public ArrayList<Integer> rewardSequence;
```

    - Initialisation :

```
display=new AgentFrame(this);

actionSequence=new ArrayList<Integer>();
stateSequence=new ArrayList<Integer>();
rewardSequence=new ArrayList<Integer>();
```

Introduction à l'Intelligence Artificielle

- **L'agent : Apprentissage des Q-values**

  - 3 ArrayList pour stocker les états, les récompenses et les actions
    - Réinitialisation pour une nouvelle partie :

```
public void initialize(){
    stateIndex=-1;
    actionIndex=-1;

    actionSequence.clear();
    stateSequence.clear();
    rewardSequence.clear();
}
```

    - Dans update, on enregistre les valeurs après avoir choisi l'action :

```
int actionIndex = (int)(Math.random()*possible_actions.size());

// update sequences
actionSequence.add(actionIndex);
stateSequence.add(stateIndex);
rewardSequence.add(reward);

return possible_actions.get( actionIndex );
```

Introduction à l'Intelligence Artificielle

- **L'agent : Apprentissage des Q-values**

  - Fonction *learn*

    - Il faut mettre à jour les Q-values à partir de notre séquence

$$Q(s_t, a_t) \Leftarrow (1-\alpha).Q(s_t, a_t) + \alpha.(r_{t+1} + \gamma . \max_{a_{t+1} \in A}(Q(s_{t+1}, a_{t+1})))$$

    - Pour la dernière étape *n* de la séquence, on prend :
      - $\max Q_{n+1} = 0$
      - r_{n+1} = reward

    - Puis, pour chaque étape k de *n-1* à *1*:
      - $\max Q_{k+1} \leftarrow \max_{ak}(Q(s_k, a_k))$
      - $r_{k+1} \leftarrow r_{k+2}$

    - On n'oubliera pas les coefficients !

Introduction à l'Intelligence Artificielle

- **L'agent : Apprentissage des Q-values**

  - Fonction *learn*

    - On déclare les coefficients dans la classe Agent :

```
public float learnRate=0.01f;
public float gamma=0.9f;
public float epsilon=1;
```

On en aura besoin pour plus tard

    - Dans la fonction *learn*, on prépare la boucle pour lire la séquence

```
public void learn(int state, int reward){

    // last step
    float maxQ=0;
    float reward_next=reward;

    for (int i=actionSequence.size()-1;i>=0;i--){

        int stateId=stateSequence.get(i);
        int actionId=actionSequence.get(i);
    }
```

Introduction à l'Intelligence Artificielle

- **L'agent : Apprentissage des Q-values**
  - Fonction *learn*
    - Écrivez ensuite la mise à jour de la Q-value

$$Q(s_t, a_t) \Leftarrow (1-\alpha).Q(s_t, a_t) + \alpha.(r_{t+1} + \gamma.\max_{a_{t+1} \in A}(Q(s_{t+1}, a_{t+1})))$$

```
int stateId=stateSequence.get(i);
int actionId=actionSequence.get(i);

Qtable.get(stateId)[actionId]= (1-learnRate) * Qtable.get(stateId)[actionId]
                               + learnRate  * ( reward_next + gamma*maxQ ) ;
```

    - Avant de passer à l'étape précédente de la séquence...

Introduction à l'Intelligence Artificielle

- **L'agent : Apprentissage des Q-values**

  - Fonction *learn*

    - … Il faut mettre à jour la récompense et le maxQ :

```
Qtable.get(stateId)[actionId]= (1-learnRate) * Qtable.get(stateId)[actionId]
            + learnRate  * ( reward_next + gamma*maxQ ) ;

// maxQ for previous step
maxQ=-1000;
for (int j=0;j<Qtable.get(stateId).length;j++){
    if (Qtable.get(stateId)[j]>maxQ) maxQ=Qtable.get(stateId)[j];
}

// read reward for previous step
reward_next=rewardSequence.get(i);
```

Introduction à l'Intelligence Artificielle

- **L'agent : Apprentissage des Q-values**

  - Fonction *learn*

    - Dans Main, on applique la fonction *learn* quand on termine une partie (pour les deux joueurs)

```
// detect end of game by victory
if (board.player1Wins()){
    System.out.println("Player 1 wins");
    gameover=true;

    player1.learn(board.getStatePlayer1(), 100);
    player2.learn(board.getStatePlayer2(),-100);
}
else{
    // detect end of game by draw
    board.getActionsPlayer2();
    if (board.actions.size()==0){
        System.out.println("draw");
        gameover=true;

        player1.learn(board.getStatePlayer1(), 10);
        player2.learn(board.getStatePlayer2(), 10);
    }
}
```

On inverse pour le joueur 2

Introduction à l'Intelligence Artificielle

- **L'agent : Apprentissage des Q-values**

  - La Q-table est maintenant prête et fonctionnelle. Nous allons afficher son contenu pour suivre son évolution : on complète la fonction *paintComponent* de *AgentPanel*

```
public void paintComponent(Graphics g){

    g.setColor(Color.white);
    g.fillRect(0,0, this.getWidth(), this.getHeight());

    if (agent.stateIndex!=-1){
        g.setColor(Color.red);
        g.fillRect(5, 15+15*agent.stateIndex, 620, 12); // current state
    }

    g.setColor(Color.black);
    for (int i=0;i<14;i++){     // labels of action index
        g.drawString(""+i, 80+40*i, 10);
    }
    for (int i=0;i<agent.states.size();i++){ // lines of the Q-table
        g.drawString(""+agent.states.get(i), 10, 25+15*i);

        for (int j=0;j<agent.actions.get(i).length;j++){
            g.drawString(""+(float)Math.round(agent.Qtable.get(i)[j]*100)/100,
                         80+40*agent.actions.get(i)[j], 25+15*i);
        }
    }
}
```

Introduction à l'Intelligence Artificielle

- **L'agent : Apprentissage des Q-values**

  - Dans la classe Agent, on ajoute un repaint( ) juste après avoir obtenu l'index de l'état actuel dans update et à la fin de la fonction learn

```
// get new state index
stateIndex=states.indexOf(state);

display.repaint();

// update sequences
```

```
    // read reward for previous step
    reward_next=rewardSequence.get(i);
}

display.repaint();
```

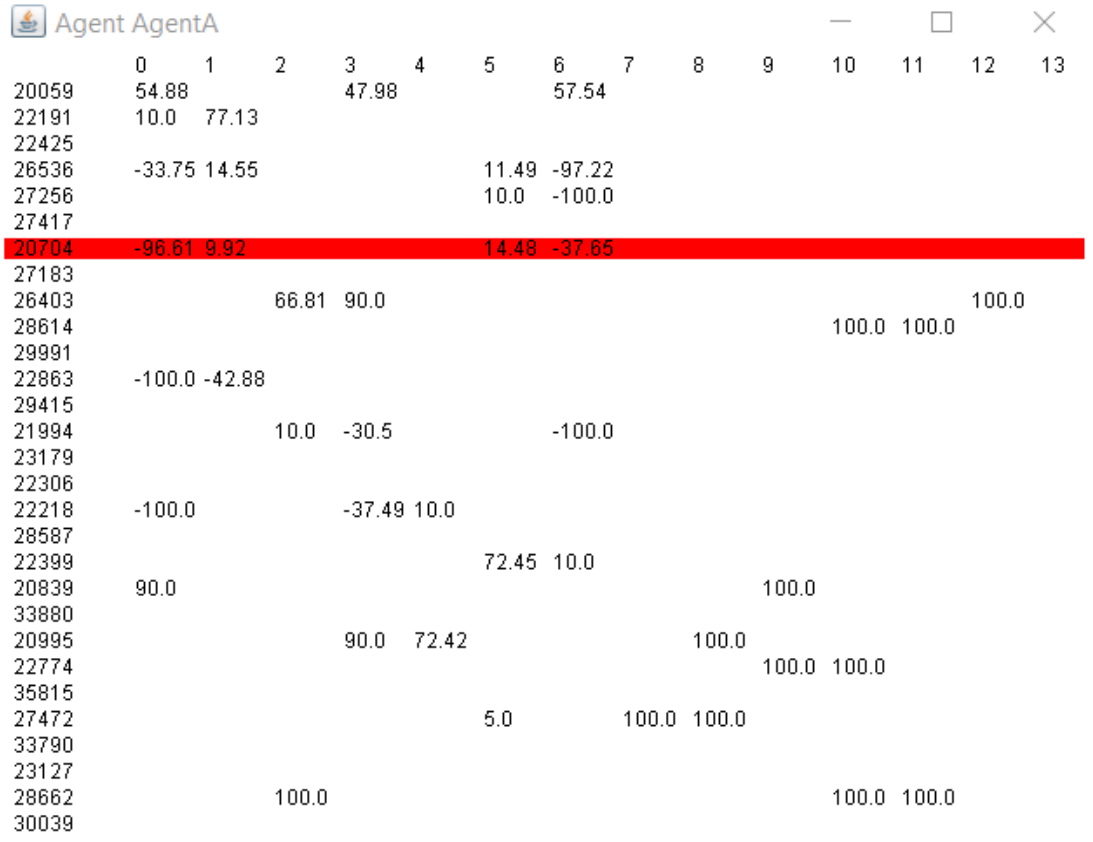
  - Dans Main, on peut créer un joueur de type Agent

```
////////////////////////////////////////////////
// select two players

player1=new Agent("AgentA", 1, this);
//player1=new RandomPlayer("AgentA", 1, this);
//player1=new Joueur("AgentA", 1, this);

player2=new RandomPlayer("AgentB", 2, this);
//player2=new Joueur("AgentB", 2, this);
```

Introduction à l'Intelligence Artificielle

- **L'agent : Apprentissage des Q-values**

  - Vous pouvez dès à présent tester votre Q-table (pour l'instant, l'agent choisi ses actions aléatoirement)



Introduction à l'Intelligence Artificielle

- **L'agent : sélection de l'action**

  - Une politique efficace : le ε-greedy

    - On commence par préparer le sélecteur dans la fonction *update* :

```
// get new state index
stateIndex=states.indexOf(state);

display.repaint();

// selection of action : epsilon-greedy
double rand=Math.random();
if (rand<epsilon){  // random selection
    actionIndex=(int)(Math.random()*possible_actions.size());
    System.out.println("random");
}
else{               // max value
    actionIndex=(int)(Math.random()*possible_actions.size());
    System.out.println("max");
}
```

Introduction à l'Intelligence Artificielle

- **L'agent : sélection de l'action**

  - Une politique efficace : le ε-greedy

    - On va diminuer progressivement le ε tous les 100 décisions, à partir de 5000 décisions
      → Il faut donc compter les décisions.
    - Déclarez une variable pour compter les décisions dans la classe agent

```
public int nbStep=0;

private AgentFrame display;
```

    - Puis on ajoute la mise à jour du ε après la prise de décision

```
// update epsilon
if (nbStep>5000 && nbStep%100==0){
    epsilon=0.99f*epsilon;
}
nbStep++;


// update sequences
```

Introduction à l'Intelligence Artificielle

- **L'agent : sélection de l'action**

  - Une politique efficace : le ε-greedy

    - On enlève les prints et on remplace la seconde partie de la condition pour chercher l'action avec la plus forte Q-value

```
// selection of action : epsilon-greedy
double rand=Math.random();
if (rand<epsilon){  // random selection
    actionIndex=(int)(Math.random()*possible_actions.size());
}
else{               // max value
    float maxval=-1000;
    for (int i=0;i<Qtable.get(stateIndex).length;i++){
        if (Qtable.get(stateIndex)[i]>maxval){
            maxval=Qtable.get(stateIndex)[i];
            actionIndex=i;
        }
    }
}
```

Introduction à l'Intelligence Artificielle

- **L'agent : sélection de l'action**

  - Une politique efficace : le ε-greedy

    - Testez votre modèle : qu'observez-vous sur les résultats des parties ?

    - Testez également les configurations random1 VS agent2 et agent1 VS agent2

```
////////////////////////////////////////////////
// select two players

player1=new Agent("AgentA", 1, this);
//player1=new RandomPlayer("AgentA", 1, this);
//player1=new Joueur("AgentA", 1, this);

player2=new Agent("AgentB", 2, this);
//player2=new RandomPlayer("AgentB", 2, this);
//player2=new Joueur("AgentB", 2, this);
```

Introduction à l'Intelligence Artificielle

- **L'agent : sauvegarde des agents**

  - Enregistrement/chargement d'un agent
    - Afin de sauvegarder une IA et pouvoir jouer contre elle, des fonctions load et save vous sont fournies.
    - Ajoutez ces fonctions dans la classe Agent (ajoutez les imports via le correcteur)
    - Ajoutez une variable PATH pour indiquer l'emplacement d'un fichier pour sauvegarder vos agents

```
public static String PATH="C:\\chemon\\vers\\dossier\\de\\sauvegarde\\";
```

    - Appelez la fonction loadFile( ) dans le constructeur de l'agent

```
        actionSequence=new ArrayList<Integer>();
        stateSequence=new ArrayList<Integer>();
        rewardSequence=new ArrayList<Integer>();

        loadFile();
    }
```

    - Le bouton save de l'interface permet maintenant de sauvegarder les joueurs de type Agent.
      - À noter : le nom du fichier sauvegardé/chargé est celui de votre agent

Introduction à l'Intelligence Artificielle

- **L'agent : sélection de l'action, le retour**

  - Nous avons défini une politique de type déterministe
  - Testons maintenant avec une politique stochastique

  - Commentez la partie sélection du max dans la fonction update

```
// selection of action : epsilon-greedy
double rand=Math.random();
if (rand<epsilon){  // random selection
    actionIndex=(int)(Math.random()*possible_actions.size());
}
else{               // max value
    /*float maxval=-1000;
    for (int i=0;i<Qtable.get(stateIndex).length;i++){
        if (Qtable.get(stateIndex)[i]>maxval){
            maxval=Qtable.get(stateIndex)[i];
            actionIndex=i;
        }
    }*/
}
```

Introduction à l'Intelligence Artificielle

- **L'agent : sélection de l'action, le retour**

  - On commence par définir les probabilités des actions avec un *softmax* sur les Q-values

$$\sigma(x_k) = \frac{e^{x_k}}{\sum_i e^{x_i}}$$

  - On créé un vecteur pour calculer et enregistrer les softmax des Q-values

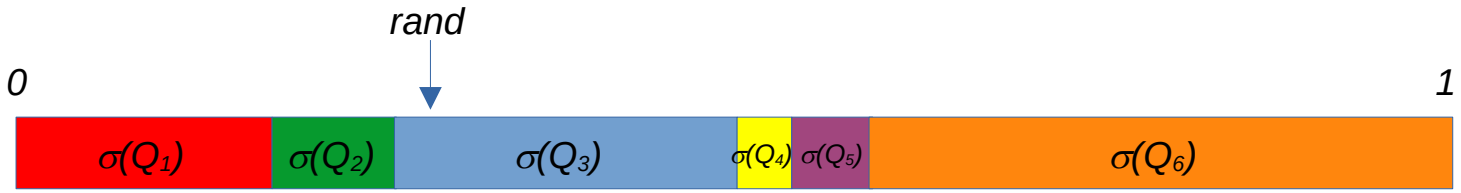
```
// compute softmax probability values
double[] softmax=new double[Qtable.get(stateIndex).length];
double sum=0;
for (int i=0;i<Qtable.get(stateIndex).length;i++){
    softmax[i]=Math.exp(Qtable.get(stateIndex)[i]);
    sum+=softmax[i];
}
for (int i=0;i<Qtable.get(stateIndex).length;i++){
    softmax[i]=softmax[i]/sum;
}
```

Introduction à l'Intelligence Artificielle

- **L'agent : sélection de l'action, le retour**

  - On tire ensuite un nombre *rand* entre 0 et 1, et on prend le nombre k pour lequel

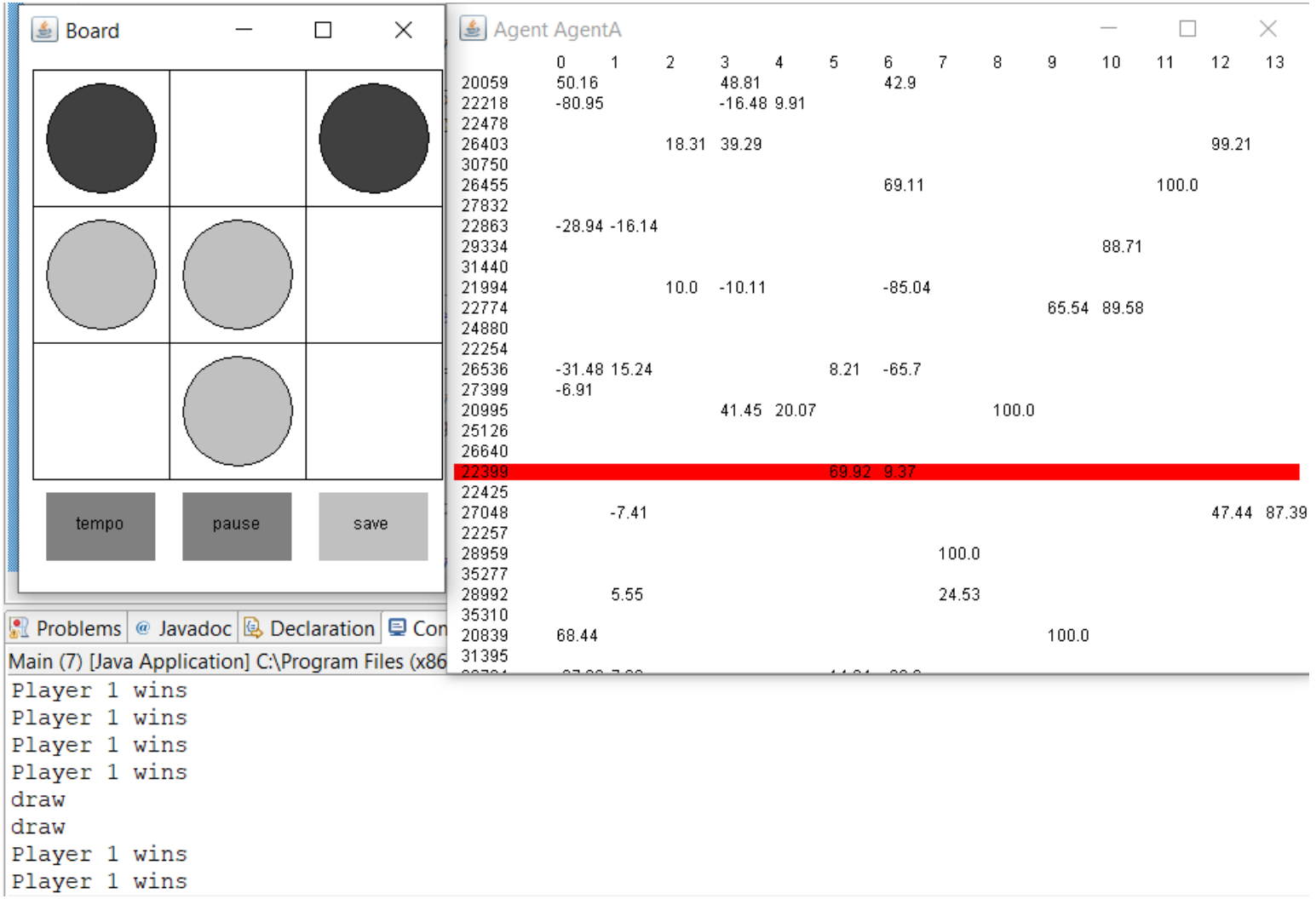
$$\sum_{1}^{k-1} \sigma(Q_k) < rand \quad \wedge \quad \sum_{1}^{k} \sigma(Q_k) \geq rand$$




```
// get a random value
double randvalue=Math.random();

// get the element corresponding to this random value
int argmax=0;
sum=softmax[0];
while (sum<randvalue){
    argmax++;
    sum+=softmax[argmax];
}

actionIndex=argmax;
```

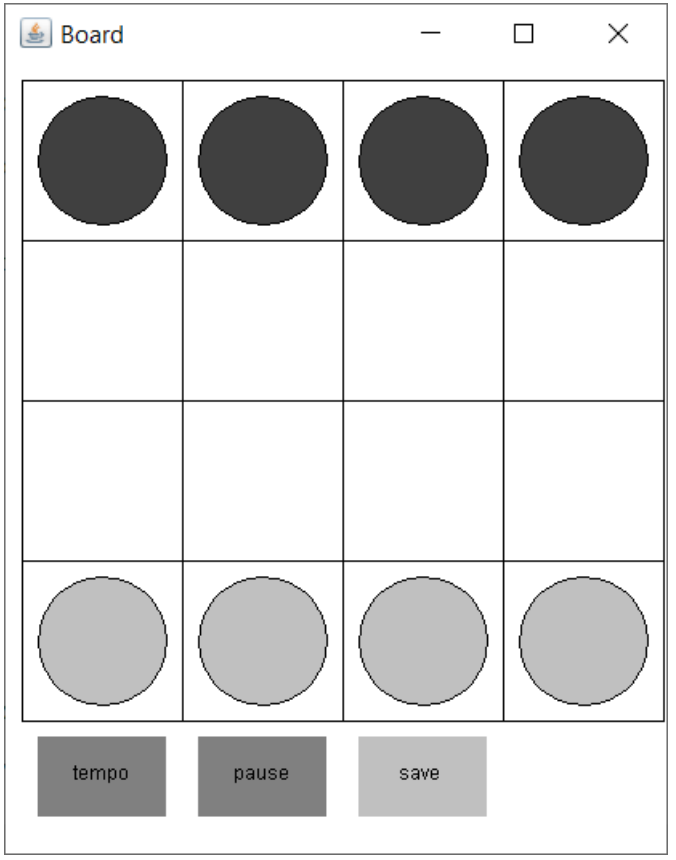
Introduction à l'Intelligence Artificielle

- **L'agent : sélection de l'action, le retour**



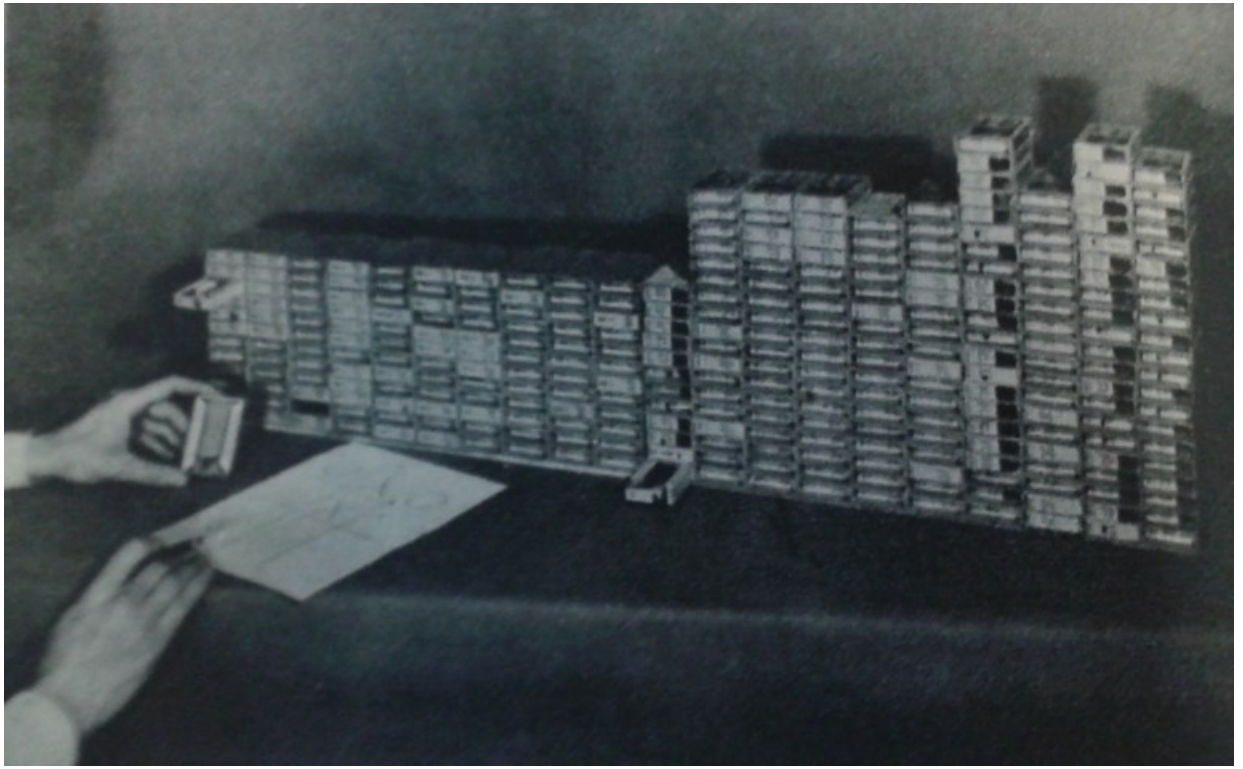
Introduction à l'Intelligence Artificielle

- **Pour aller plus loin**
  - Vous trouverez également sur le github l'environnement pour le jeu de l'octapion



Introduction à l’Intelligence Artificielle

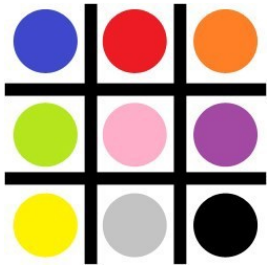
- **Pour aller plus loin**
  - MENACE (Donald Michie, 1960) : une machine qui apprend à jouer au morpion et constituée… de boites d’allumettes et de haricots



Introduction à l'Intelligence Artificielle

- **Pour aller plus loin**

  - Chaque boite correspond à une configuration (état) du jeu (en considérant les symétries et rotations, on réduit à 304 états)



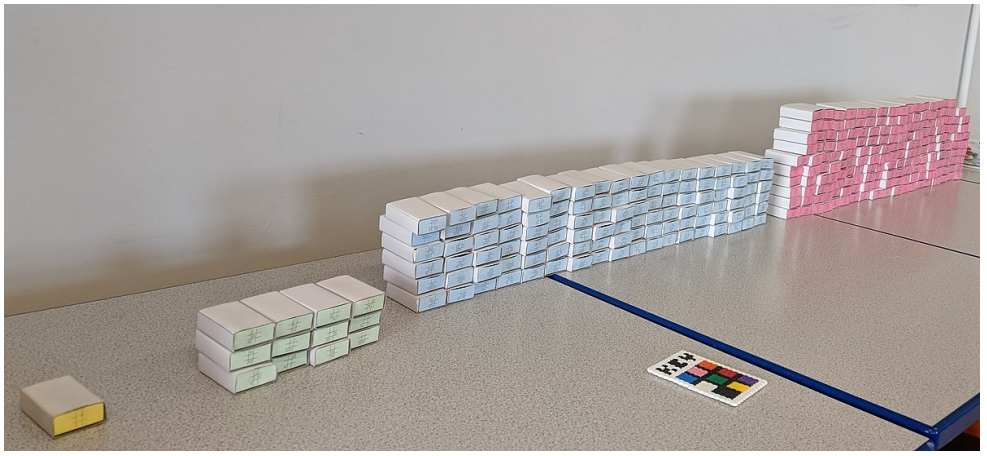
  - 9 variétés de haricots correspondant chacune à une case du plateau
    - Dans les versions plus récentes, des perles de couleur sont utilisées

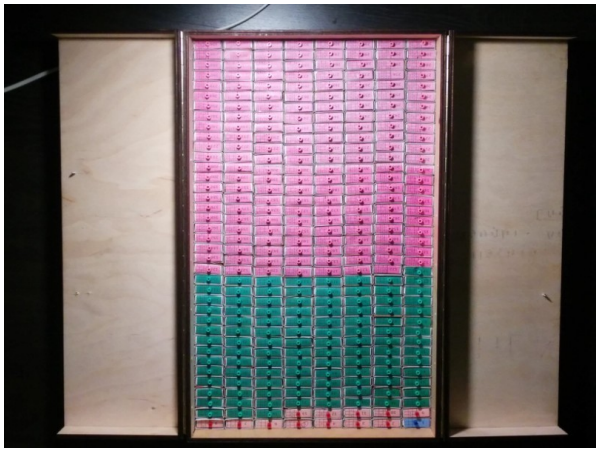
Introduction à l'Intelligence Artificielle

- **Pour aller plus loin**

  - MENACE joue en premier. Pour obtenir l'action, on utilise la boite correspondant à la situation actuelle (moyennant rotation et/ou symétrie), et on tire au hasard un haricot de cette boite
    - Il s'agit donc d'une politique de type stochastique !

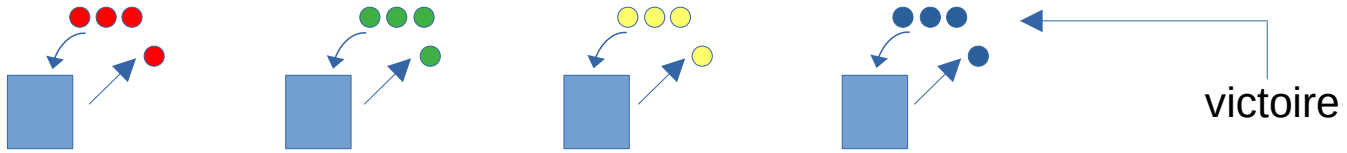
  - Le joueur joue à son tour, puis on répète l'opération



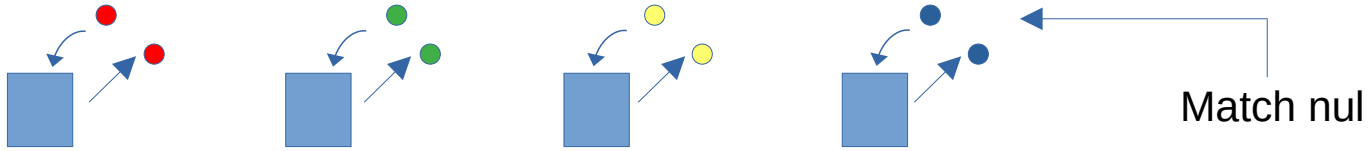


Introduction à l'Intelligence Artificielle

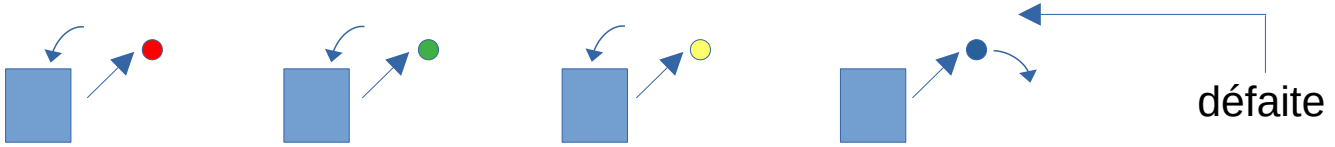
- **Pour aller plus loin**
  - Une fois la partie terminée, on applique les changements suivants :
    - Si MENACE a gagné, on ajoute 3 haricots du type tiré dans chaque boites → renforcement du chemin menant à la victoire



    - Si match nul, on ajoute 1 haricot du type tiré dans chaque boite
      → renforcement plus faible



    - En cas de défaite, on retire le haricot tiré de la dernière boite utilisée.
      - Si la boite est désormais vide, on retire également le haricot précédent (et ce, de façon récursive)
        → affaiblissement et/ou suppression des chemins menant à une défaite



Introduction à l’Intelligence Artificielle

- **Pour aller plus loin**
  - Un tutoriel pour construire ce genre de ‘machine’ pour l’hexapion :

    https://www.instructables.com/Matchbox-Mini-Chess-Learning-Machine/