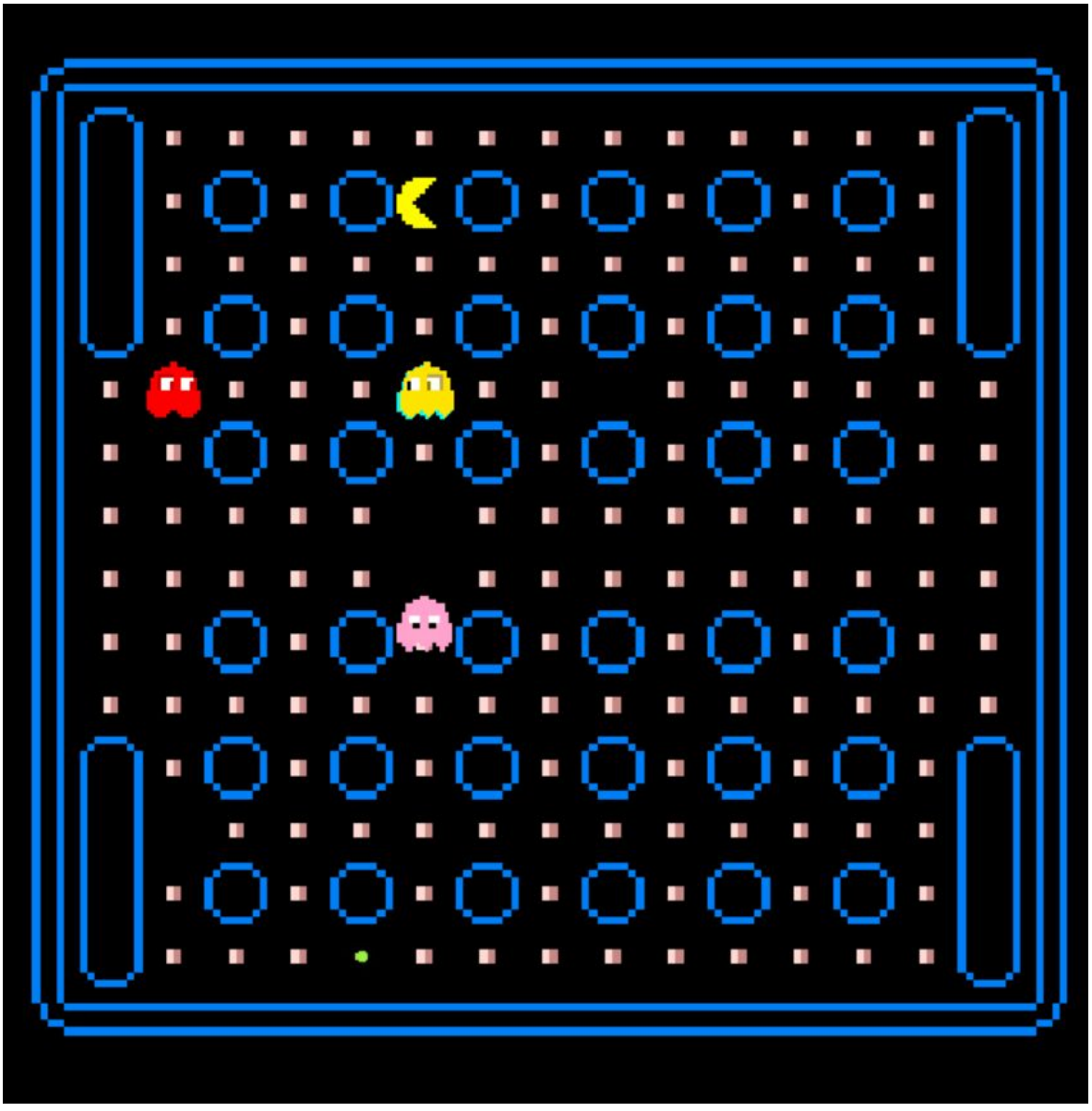
Projet de Mathématiques, Probabilités,
Mémoire

Yoann KOEPPEL, Pierre THIEL

PACMAC



Lien de téléchargement……..: Version 0.34
Lien du dépôt……: Pacmac

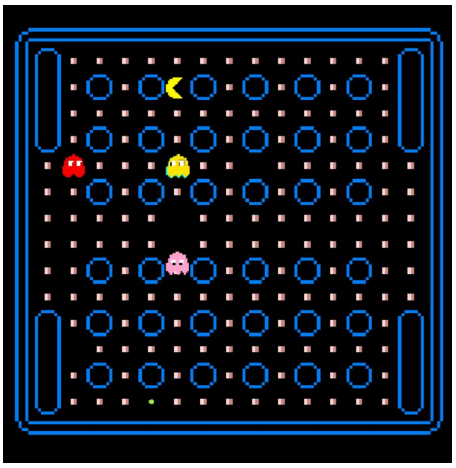
1. Le jeu

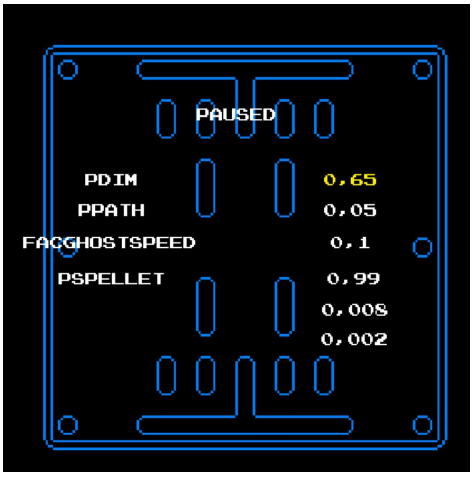
Pacmac est une refonte du jeu *Pacman* original, sorti en 1980, dans un style procédural. En effet, le jeu original ne contient qu'un seul niveau, et les objets (*fruits*) et adversaires (*fantômes*) possèdent des comportements fixés. Ils nous semblent ainsi intéressant de faire varier:

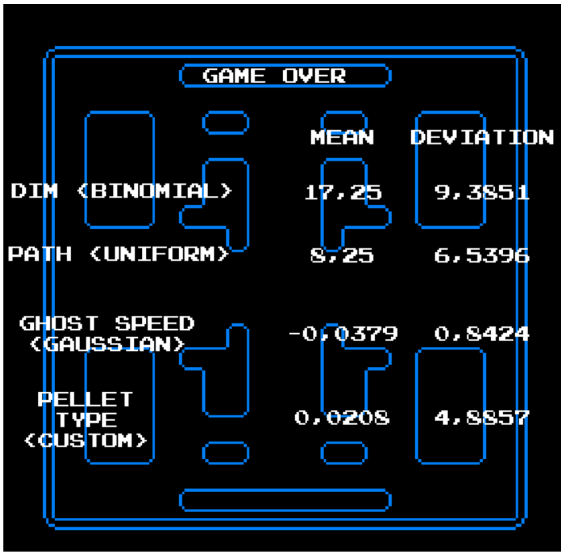
1) Les dimensions du niveau
2) et 3) Les chemins empruntables par les fantômes et Pacman (nombre et position)
4) La vitesse des fantômes
5) Le type des fruits

Le jeu est divisé en trois parties :

1) Les niveaux, qui sont joués
2) Le menu, dans lequel le joueur peut modifier les lois aléatoires
3) L'écran de fin, dans lequel il consulte les résultats des tirages.







2. Les variables aléatoires

Représentation

Remarquons tout d'abord qu'on peut modéliser une probabilité par un réel appartenant à $[0;1]$. Le sujet autorise donc à travailler à partir d'une probabilité piochée de manière uniforme sur cet intervalle.

Il s'agit ensuite de trouver $A \subset \Omega$ qui vérifie $p = P(X \in A)$, avec p cette probabilité. On a donc un problème d'**existence** et un problème d'**unicité** de A.

On choisit $A = \{\omega \in \Omega \mid \omega \leq a\}$ pour un a donné.

La fonction $f : a \mapsto p = P(X \leq a)$ nous donnera donc un élément de Ω choisi aléatoirement suivant la loi de X.

En termes d'implémentation, on a une classe abstraite Distribution, avec T correspondant $Img(X)$:

```
public abstract class Distribution<T> : IDistributor<T>
{
     abstract public T Distribute(Probability prob);
    /* ... */
}
```

Toute les lois implémentées hériteront de Distributor.

Nous implémentons en particulier des variables aléatoires à image *finie totalement ordonnée* et *non finie*.

1) Le cas fini (totalement ordonné)

Soit X une variable aléatoire définie sur un univers Ω fini, à valeurs dans un ensemble fini. On peut dans ce cas modéliser $Img(X)$, et $(P(X=\omega))_{\omega\in\Omega}$ comme des tableaux de valeurs indexés par les $\omega \in \Omega$. On peut ainsi, pour tout a, connaître $P(X=a)$, puis $p = P(X \leq a)$. La fonction $f : a \mapsto p = P(X \leq a)$ est strictement croissante, à image finie, donc bijective. la bijection réciproque $f^{-1} : p \mapsto a$ permet de piocher un $a \in \Omega$ suivant la loi régissant X.

En implémentation:

```
public abstract class FiniteDistribution<T> : Distribution<T>
{
      private T[] _values;
      protected Probability[] _quantileValues;
      override public T Distribute(Probability prob)
      {
            for (int i=0; i<GetNumber(); ++i)
            {
                  if (prob <= GetQuantileValue(i))
                  {
                      /* right one */
                      return _values[i];
                  }
            }
       }
       /* ... */
}
```

note: f^{-1} *correspond en fait à la fonction de densité de probabilité cumulative inverse, ou* fonction quantile, discrète.

2) Le cas non fini

Dans ce cas, on utilise une loi connue (donc de *fonction quantile* connue) : c'est elle que l'on garde en mémoire.

note: Il s'agira de l'estimer si elle n'est pas représentable directement par les fonctions usuelles, comme c'est le cas pour la loi gaussienne *ou* normale, *où le problème revient à évaluer* $x \mapsto \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$, appelée *fonction d'erreur gaussienne*, en a.

Exemple d'implémentation avec la loi gaussienne:

```
public class GaussianDistribution : Distribution<double>
{
	private double _mu;
	private double _sigma;
	public GaussianDistribution(double mu, double sigma)
	{
		_mu = mu;
		_sigma = sigma;
	}
	override public double Distribute(Probability p)
	{
		return QuantileFunction(_mu, _sigma, p);
	}
	/* ... */
}
```

Caractéristiques

1) Dimensions du niveau

- Phénomène modélisé : Variation de la taille du niveau en deux dimensions.
- Loi choisie : Deux lois binomiales sur $\Omega \equiv [min; max]$, paramétrées par l'indice du niveau. On aurait aussi pu utiliser une unique loi multinomiale dans $\Omega \equiv [min; max]^2$.
- Simulation : voir implémentation des lois finies. Calcul du binôme de Newton, puis remplissage des valeurs de la fonction quantile :

```
public class BinomialDistribution : FiniteRangeIntDistribution
      {
      private Probability _p;
      public BinomialDistribution(Probability p, int min, int max)
      : base(min, max)
      {
            _p = new Probability(p);
            PopulateQuantileValues();
      }

      private void PopulateQuantileValues()
      {
            for (int k=0; k<GetNumber()-1; ++k)
            {
                  _quantileValues[k] =
(double)((Double)MathsUtil.BinomialCoefficient(k, GetNumber())
                  * Math.Pow(_p.GetValue(), k)
                  * Math.Pow(_p.GetInverseEventProb().GetValue(),
GetNumber() - k));
                  if (k > 0)
                  {
                  _quantileValues[k] += _quantileValues[k - 1];
                  }
            }
            _quantileValues[GetNumber()-1] = 1.0;
      }
      /* ... */
}
```

- Choix des paramètres : le maximum des dimensions varie à la racine carré du niveau : les niveaux auront tendance à s'agrandir. Le choix de p est laissée au joueur, pour pouvoir faire varier l'écart-type et avoir des niveaux plus ou moins égaux.

2) Nombre de chemins

- Phénomène modélisé : Nombre de chemins tracé dans le niveau.
- Loi choisie : Loi hypergéométrique sur $\Omega \equiv [min; max]$
- Simulation : voir implémentation des lois finies. Calcul des binômes de Newton etc.

```
public class HypergeometricDistribution : FiniteRangeIntDistribution
      {
      private int _N;
      private Probability _p;
      public HypergeometricDistribution(Probability p, int min, int max,
int N)
      : base(min, max)
      {
                /* ... */
                PopulateQuantileValues();
      }
      private void PopulateQuantileValues()
      {
            for (int k=0; k<GetNumber()-1; ++k)
            {
                  _quantileValues[k] =
(double)(MathsUtil.BinomialCoefficient(
                  (int)(_N * _p.GetValue()), k)
                  * MathsUtil.BinomialCoefficient(
                  (int)(_N * _p.GetInverseEventProb().GetValue()),
                  GetNumber() - k)
                  / MathsUtil.BinomialCoefficient(_N, GetNumber())
                  );
                  if (k > 0)
                  {
                        _quantileValues[k] += _quantileValues[k - 1];
                  }
            }
            _quantileValues[GetNumber()-1] = 1.0;
        }
        /* ... */
}
```

- Choix des paramètres : Le minimum est de l'ordre du minimum des dimensions du niveau, le maximum de la moyenne, permettant de contenir le nombre de chemins.

- Structure de corrélation : le minimum et maximum dépend de la dimension du niveau.

3) Position des chemins

- Phénomène modélisé : Position des « coudes » reliant les chemins.
- Loi choisie : Deux lois uniformes sur $\Omega \equiv [0;x] \times [0;y]$, où $(x;y)$ sont les dimensions du niveau. On aurait aussi pu utiliser une loi paramétrée pour modifier la dispersion des coudes.

- Simulation : voir implémentation des lois finies.
- Structure de corrélation : idem, Ω dépend des dimensions du niveau.

4) Vitesse des fantômes

- Phénomène modélisé : Vitesse de déplacement des fantômes dans le niveau.
- Loi choisie : Loi gaussienne. La vitesse des fantôme étant positive, on prend en fait la valeur absolue de la loi.
- Simulation : voir implémentation des lois non finies. On utilise ici l'approximation de la fonction quantile par Shore (1980) :
-

$$f^{-1}(p) \approx 5.5556[1 - (\frac{p}{1-p})^{0,1186}], p \geq 1/2$$

$$f^{-1}(p) \approx -5.5556[1 - (\frac{p}{1-p})^{0,1186}], sinon$$

```
public static FloatType
ShoreStandardNormalQuantileFunction<FloatType>(random.Probability p)
            where FloatType : unmanaged
      {
            double r, epsilon;
            bool condition = p.GetValue() >= 0.5;
            r = condition ? p : 1 - p;
            epsilon = condition ? 1.0 : 0.0;
            return (FloatType)(object)(epsilon * 5.5556*(1 - Math.Pow(r
/ (1 - r), 0.1186)));
      }
```

- Choix des paramètres : L'utilisateur contrôle l'espérance $\mu \in [0;1]$.

5) Type des fruits

- Phénomène modélisé : Type des fruits qui doivent apparaître.
- Loi choisie : Loi arbitraire sur $\Omega \equiv \{Pacdot; SuperPellet; PowerPellet\}$.
- Simulation : voir implémentation des lois non finies. On met Ω en bijection avec $\{0; 1; 2\}$.
- Choix des paramètres : L'utilisateur contrôle la probabilité d'apparition des fruits $(p1; p2; p3)$.

3. Difficultés

Le jeu *Pacmac* possède de nombreux éléments avec lesquels il est facile d'imaginer un fonctionnement aléatoire. Dans les conditions de cette exercice, seule la compréhension des lois et leur implémentation aurait pu nous poser des difficultés.