Exploration

Objectifs

Un gradin est représenté par une grille dont les cellules indiquent si les sièges sont occupées ou non. Le nombre de spectateurs est fixé.

Les objectifs sont:

- de diagnostiquer l'espacement entre les spectateurs pour comparer les configurations,
- de simuler une configuration optimale.

Données

La grille est une matrice binaire, vraie quand un spectateur est assis. La taille et le nombre de spectateurs sont paramétrés.

```
# Paramétrage : taille
num_rows <- 10
num_cols <- 15
# Nombre de spectateurs
num_cells <- num_rows * num_cols /2
```

Une configuration de départ est tirée aléatoirement.

Technique : mélanger deux vecteurs de `TRUE` et `FALSE` puis trasformer le vecteur obtenu en matrice. Une matrice est aussi un vecteur. Les éléments sont rangés par colonnes, en commençant par le coin haut gauche.

```
library("magrittr")
sample(c(rep(TRUE, num_cells), rep(FALSE, num_rows * num_cols - num_cells))) %>%
  matrix(nrow = num_rows) ->
  seats
```

Carte

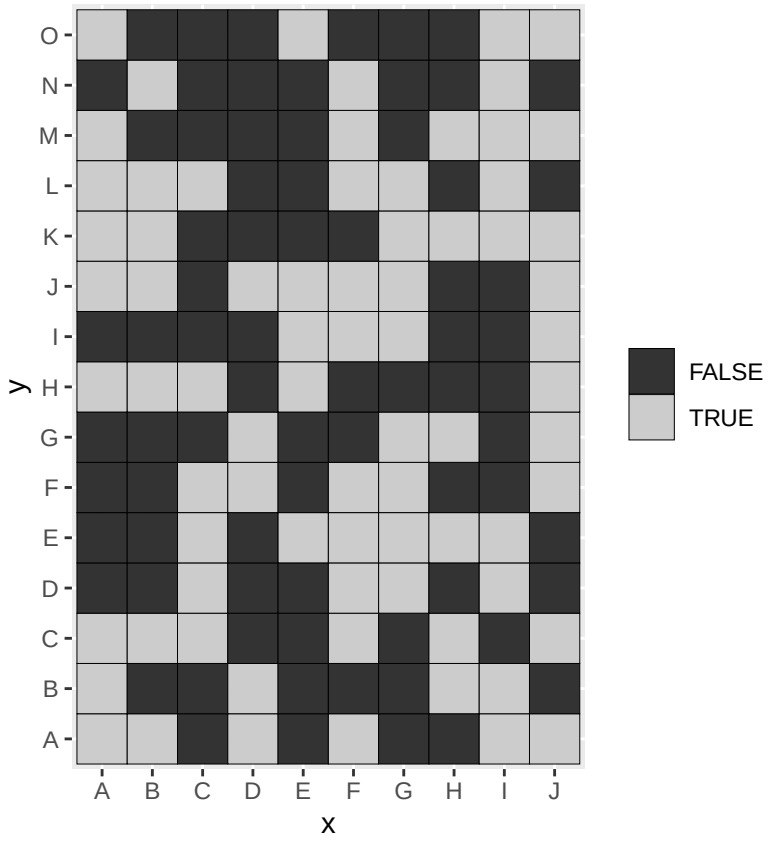
La carte des sièges est dessinée par la fonction `plot_map()`.

A faire : la carte est mal orientée, les lignes sont verticales, les colonnes horizontales. Le premier siège est (A, A), le deuxième à sa droite, puis de gauche à droite et de bas en haut.

```
library("ggplot2")
plot_map <- function(the_pattern) {
  the_df <- as.data.frame.table(the_pattern)
  names(the_df) <- c("x", "y", "z")
  the_plot <- ggplot(data=the_df, ggplot2::aes_(x=~x, y=~y)) +
    geom_tile(aes_(fill=~z), col="black") +
    theme(panel.grid = ggplot2::element_line()) +
    coord_fixed() +
    labs(fill=unique(the_pattern))+
    theme(legend.title = element_blank())+
    scale_fill_grey()
```

```
  return(the_plot)
}

seats %>% plot_map
```




```
# Occupation des premiers sièges (ligne du bas)
seats[1:10]
```

```
##  [1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE
```

Promiscuité

Le nombre de voisins moyen d'un spectateur est l'indicateur de promiscuité. L'objectf est de le minimiser.

Le nombre de voisins d'un siège est calculé par `num_neighbors()`. `seats` est la matrice. `seat` est l'indice du siège dans le vecteur équivalent à la matrice.

Technique : repérer un siège par son indice dans le vecteur est plus efficace que par ses coordonnées dans la matrice. Pour trouver ses voisins, il faut passer de cet indice aux coordonnées. `seats[seat]` vaut 1 si le siège est occupé (`TRUE` vaut 1).

```
num_neighbors <- function(seats, seat) {
  # From vector to matrix. Size
  num_rows <- nrow(seats)
  num_cols <- ncol(seats)
  # Coordinates of the seat
  seat_row <- (seat - 1) %% num_rows +1
  seat_col <- (seat - 1) %/% num_rows + 1
```

```
  # Window around the seat
  min_row <- max(seat_row -1, 1)
  max_row <- min(seat_row +1, num_rows)
  min_col <- max(seat_col -1, 1)
  max_col <- min(seat_col +1, num_cols)
  # Return the number of occupied seats, minus 1 if the seat itself is occupied
  return(sum(seats[min_row:max_row, min_col:max_col]) - seats[seat])
}
```

Le nombre moyen de voisins `avg_num_neighbors()` est la fonction diagnostique de la simulation.

Technique : `which(seats)` retourne le vecteur des indices des sièges occupés. `sum(seats)` retourne le nombre de sièges occupés.

```
avg_num_neighbors <- function(seats) {
  # Initialize the number of neighbors
  num_nb <- 0
  for (seat in which(seats)) {
    # Number of neighbors of each occupied seat
    num_nb <- num_nb + num_neighbors(seats, seat)
  }
  return(num_nb / sum(seats))
}

# Average number of neighbors
avg_num_neighbors(seats)
```

```
## [1] 3.386667
```

Mise à distance

Algorithme de Metropolis-Hastings

L'approche des processus de Strauss peut être utilisé dans la grille.

Chaque voisinage de points a un coût, par exemple 1 pour chaque voisin immédiat d'un spectateur (comme dans un processus de Gibbs). La probabilité d'une configuration est définie à une constante près relativement à la configuration de départ. Le rapport de probabilités entre deux configurations (Intensité conditionnelle de Papangelou) est le rapport des exponentielles négatives de leurs coûts. Avec le coût choisi, si le passage d'une configuration à une autre modifie le nombre total de voisins de d_n, alors leur rapport de probabilité est $\lambda^\star = e^{-d_n]}$. La nouvelle configuration est plus probable ($\lambda^\star > 1$) si le nombre de voisins diminue ($d_n < 1$).

Le coût peut être ajusté (multiplié par une constante) pour que la probabilité varie plus ou moins quand un voisin est ajouté ou supprimé.

L'algorithme pour simuler le processus qui minimise le coût est celui de Metropolis-Hastings. C'est un algorithme MCMC (Monte Carlo Markov Chains):

- Sélectionner un siège occupé

```
from_seat <- sample(which(seats), size=1)
```

- Sélectionner un siège libre

```
to_seat <- sample(which(!seats), size=1)
```

- Calculer la différence du nombre de voisins $d - n$

```
d_n <- num_neighbors(seats, to_seat) - num_neighbors(seats, from_seat)
```

- Comparer $e^{-adjust \times d_n}$ à un nombre aléatoire, tiré entre 0 et 1. Le coût est ajusté pour que le taux de succès des tentatives de mouvement soit convenable. Tous les mouvements favorables (qui diminuent le nombre de voisins) sont conservés. Plus `adjust` est grand, plus il est difficile de conserver un mouvement défavorable.

```
adjust <- 5
move <- (exp(-adjust*d_n) > runif(1))
```

- Retenir le mouvement si $e^{-adjust \times d_n}$ est suffisant.

```
if (move) {
  seats[from_seat] <- FALSE
  seats[to_seat] <- TRUE
}
```

L'opération est répétée un grand nombre de fois.

Simulation

Les paramètres sont le nombre d'itérations et le facteur d'ajustement du coût.

```
iterations <- 1E5
adjust <- 10
```

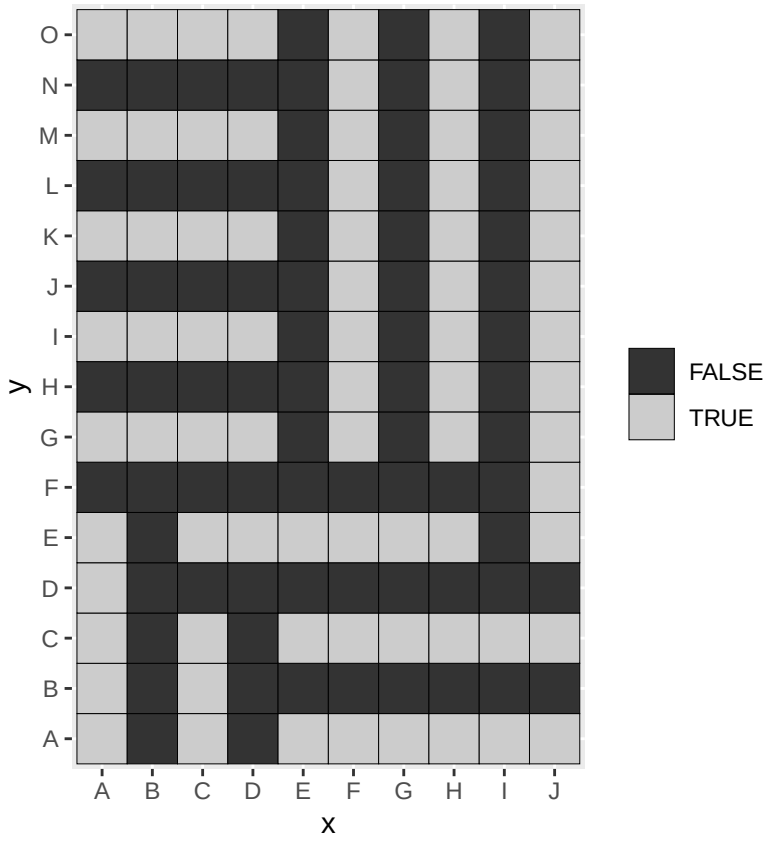
La boucle complète est la suivante:

```
num_moves <- 0
for (i in 1:iterations) {
  from_seat <- sample(which(seats), size=1)
  to_seat <- sample(which(!seats), size=1)
  d_n <- num_neighbors(seats, to_seat) - num_neighbors(seats, from_seat)
  move <- (exp(-adjust*d_n) > runif(1))
  if (move) {
    seats[from_seat] <- FALSE
    seats[to_seat] <- TRUE
    num_moves <- num_moves + 1
  }
}

# Number of actual moves
num_moves
```

```
## [1] 223
```

```
# Plot the configuration
seats %>% plot_map()
```




```
# Average number of neghbors
avg_num_neighbors(seats)
```

```
## [1] 1.653333
```

La configuration finale affiche moins de voisins. Les bordures de la tribune sont très sollicités.

Si `adjust` est trop grand, les mouvements défavorables sont rarement acceptés et la configuraiton peut se bloquer dans un minimum de coût local. S'il est trop petit, tous les mouvements sont acceptés et la simulation ne converge pas.