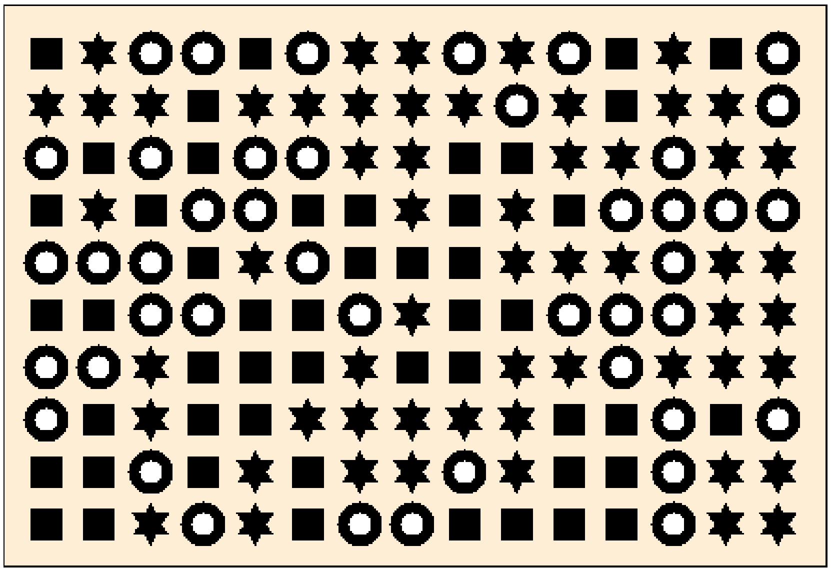
Université Paris 6
Licence d'informatique

Module de Programmation
Année 2000-2001

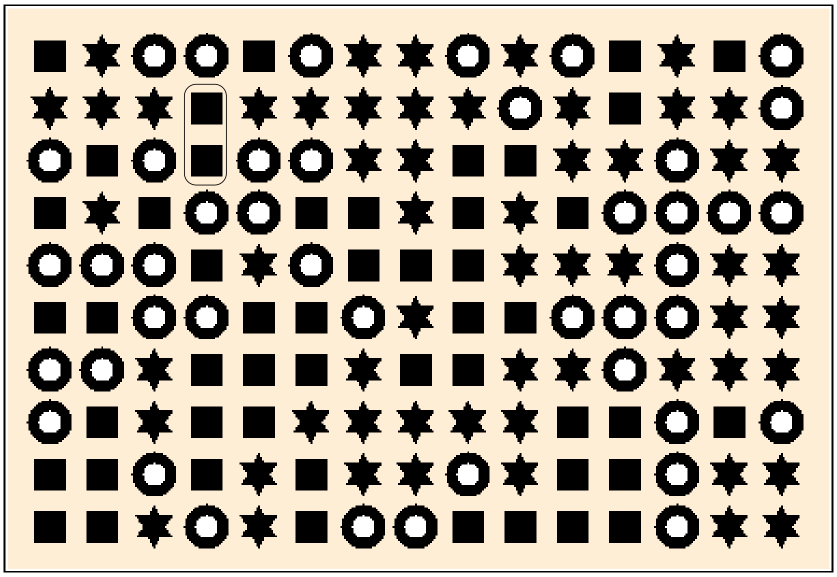
Travaux Pratiques n° 1
Le jeu « SameGame »

1 Description d'une version simple du jeu

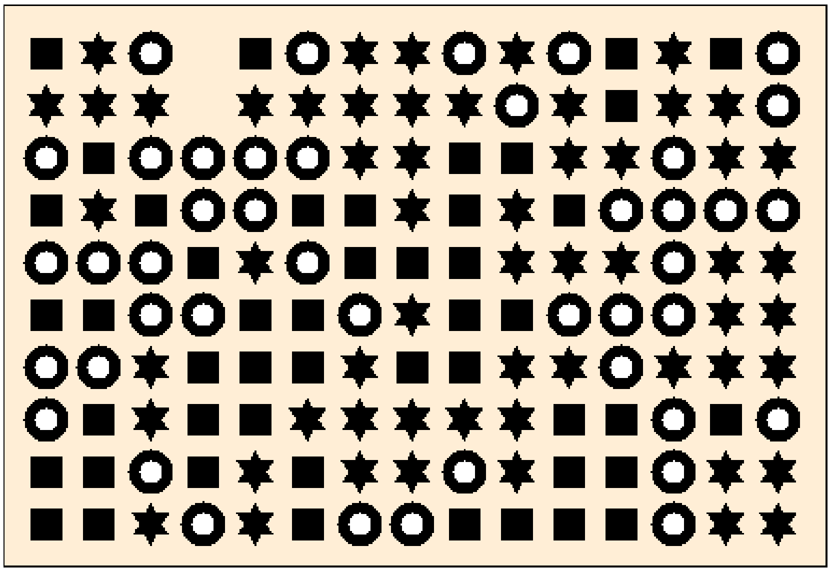
Le tableau de jeu a la forme d'un rectangle de 10 lignes et 15 colonnes où figurent des pièces variées (disques de couleur ou bien comme dans cet énoncé pièces géométriques), avec 3 types possibles. Ces pièces sont tirées au hasard au début du jeu. On peut quitter le jeu à tout instant en tapant `q` (pour quitter) dans la fenêtre graphique. Voici un exemple de tableau de jeu au début d'une partie :



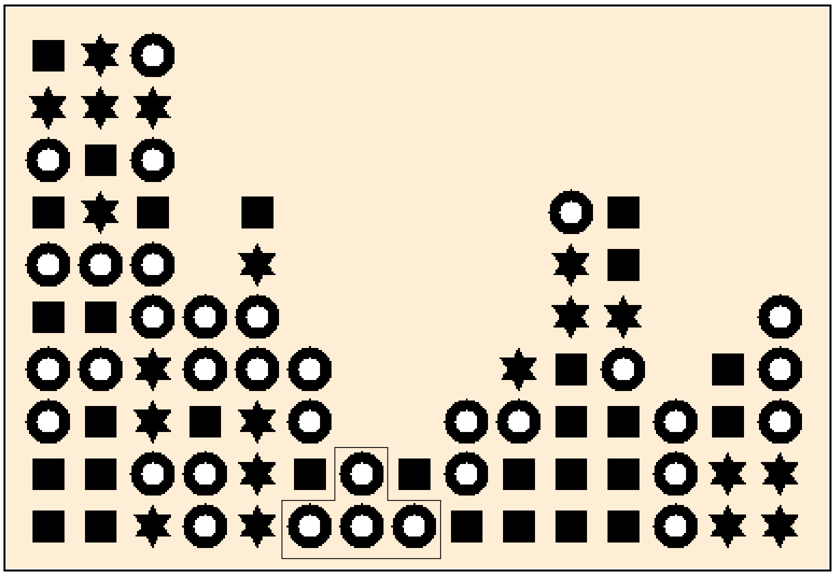
Il s'agit pour le joueur d'enlever le plus possible de pièces sur le rectangle de jeu, sachant que pour retirer une pièce il faut que cette pièce fasse partie d'une *sélection*. Une sélection comprend nécessairement au minimum deux pièces du même type disposées côte à côte verticalement ou horizontalement. La sélection s'effectue en cliquant sur une pièce et le programme détermine lui-même quel est l'ensemble des pièces qui forment une chaîne ininterrompue de sélections élémentaires à partir de la pièce désignée par le joueur. Par exemple sur le jeu ci-dessus, si l'on clique sur la pièce qui se trouve dans la huitième ligne à partir du bas et la quatrième colonne à partir de la gauche, alors la sélection effectuée correspond à la zone entourée sur la vue suivante :



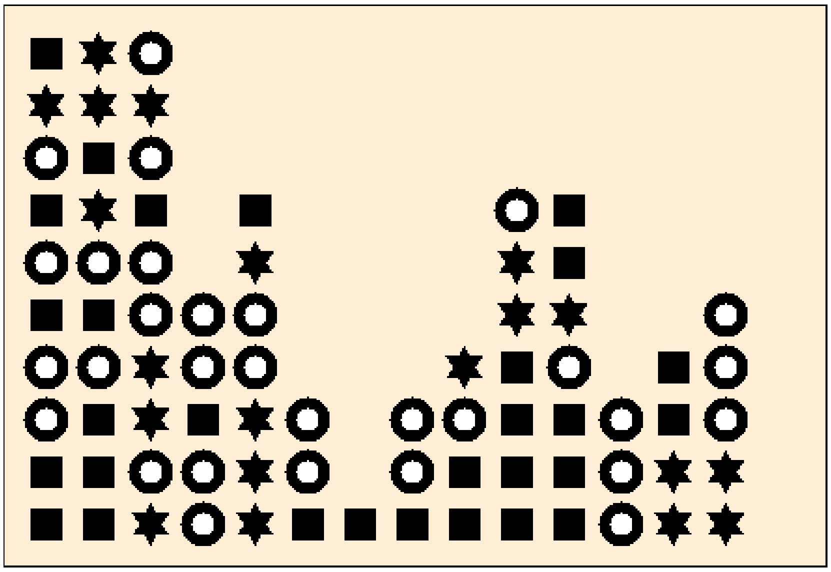
Cette sélection va disparaître selon la première règle suivante : les pièces qui se trouvaient au dessus des pièces de la sélection descendent pour prendre leur place. Pour la sélection de notre exemple, cela donne :



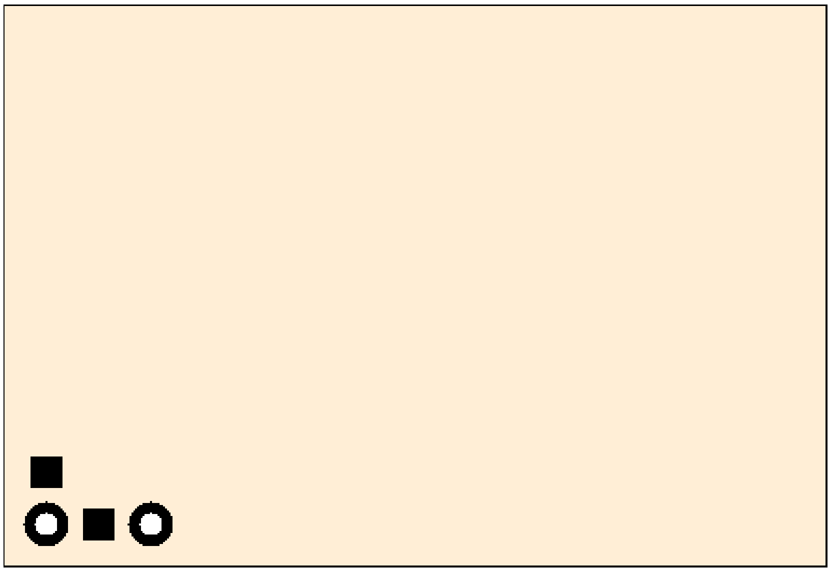
Si la suppression de la sélection produit une colonne vide, comme pour la sélection ci-dessous :



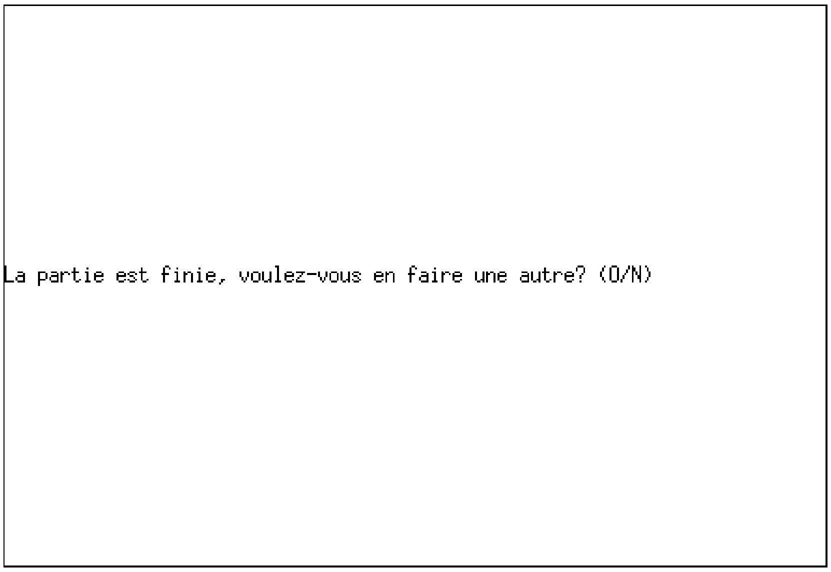
alors cette colonne vide disparaît et les colonnes suivantes sont décalées d'un cran sur la gauche, comme le montre la figure ci-dessous pour la deuxième sélection de notre exemple.



Le jeu est terminé quand il n'est plus possible de retirer aucune pièce du tableau de jeu comme dans le cas ci-dessous.



Il apparaît alors un message pour proposer une nouvelle partie ou quitter. Par exemple :



Une version complète du jeu avec historique, scores, table des meilleurs scores, visualisation de la sélection avec le nombre de pièces qu'elle contient, etc. est disponible sur `http://www-calfor.lip6.fr/~vmm/Enseignement/Licence_info/Programmation/TP/TP1/enonce_complet.html`.

2 Description du projet

2.1 Représentation interne du jeu

Tableau débordant

On représente le tableau de jeu par un tableau débordant, c'est-à-dire que si le jeu comprend ℓ lignes et c colonnes, on crée un tableau de taille $(\ell + 2) \times (c + 2)$, dont les indices de lignes peuvent varier de 0 à $\ell + 1$ et les indices de colonnes de 0 à $c + 1$.

Ceci a pour effet que, lors de la détection de la zone sélectionnée, pour tout indice de ligne i (resp. de colonne j) compris entre 1 et ℓ (resp. c), les lignes $i - 1$ et $i + 1$ (resp. les colonnes $j - 1$ et $j + 1$) du tableau existent.

On représente graphiquement les lignes (resp. colonnes) d'indice compris entre 1 et ℓ (resp. c).

Pièces

Chaque case du tableau a une couleur qui est celle de la pièce s'il y en a une ou bien celle du fond s'il n'y en a pas. Par exemple, les bords du tableau (ces lignes et colonnes d'indice 0, $\ell + 1$ et $c + 1$) sont de la couleur du fond. Ceci nous sert lors de la détection de la zone sélectionnée.

Par ailleurs, lors de cette détection, on a besoin de pouvoir marquer les cases selon que l'on sait déjà que la pièce appartient ou n'appartient pas à la zone sélectionnée, ou que cette pièce n'a pas encore été examinée.

On définit donc un type somme qui représente cette information de marquage, un type somme pour les couleurs si on le désire, puis un type produit qui contient une couleur et une marque. Les éléments du tableau sont de ce type.

2.2 L'algorithme central : la détection de la zone sélectionnée et sa suppression

Le marquage

Dans un premier temps, pour détecter la zone sélectionnée dans le tableau t à la i^{e}ligne et la j^{e}colonne, on va passer d'une situation où toutes les cases sont marquées `Peut_être` à une situation où toutes les cases sont marquées `Oui` ou `Non`. Pour cela on procède récursivement dans les quatre directions à partir de la case de départ en s'arrêtant aux bords et en essayant de ne marquer que les cases qui ne le sont pas déjà (celles marquées `Peut_être`).

La suppression

Lorsque le tableau est entièrement marqué de cette façon et que la sélection comporte au moins deux pièces, on va tout d'abord supprimer dans chaque colonne les pièces qui appartiennent à la sélection, puis supprimer les colonnes vides, démarquer les cases du tableau et mettre à jour la représentation graphique du tableau de jeu.

2.3 La bibliothèque graphique

Pour la représentation graphique, on utilisera la bibliothèque `libgraph` de CAML en l'appelant interactivement de la façon suivante :

```
camllight camlgraph
```

ou pour obtenir un exécutable dans une commande batch de la forme

```
camlc -custom unix.zo graphics.zo same_game.ml -o same -lgraph -lunix -lX11
```

Cette bibliothèque est documentée dans le manuel de référence qui vous a été distribué au début de l'année ou bien sur le Web (en allant dans le menu `Tuareg` sous `emacs` ou bien directement à `file:/usr/doc/caml-light-0.74/man-caml/index.html`l'adresse locale ou encore `http://caml.inria.fr/man-caml/index.html`le site d'origine).

À titre indicatif, voici des fonctions qui seront probablement utilisées dans votre projet :
- `open_graph`, `close_graph`, `clear_graph` pour ouvrir, fermer et effacer la fenêtre graphique ;
- `set_color`, `rgb` et les couleurs prédéfinies pour la gestion des couleurs ;
- `fill_rect` et `fill_circle` pour le coloriage ;
- `draw_string` et `draw_char` pour l'écriture ;
- `moveto` pour se déplacer.

2.4 Le hasard

Le hasard dans le choix de la couleur des pièces lorsque l'on produit le tableau de jeu sera obtenu en utilisant le module `random` de la *standard library*. Pour initialiser le générateur aléatoire, on peut utiliser la valeur fournie par la fonction `unix__time`, de la bibliothèque `unix` qui est disponible avec la bibliothèque graphique. Pendant la mise au point du programme on laissera la ligne d'initialisation du générateur en commentaires pour avoir des parties reproductibles.

2.5 La gestion évènementielle

On dispose avec la bibliothèque graphique de quelques fonctions de gestion évènementielle. Il s'agit de détecter les actions au niveau de la souris et du clavier dans la fenêtre graphique.

La souris

La seule action de la souris qui nous intéresse est d'attendre que le joueur clique sur une pièce et de connaître la position de cette pièce.

La fonction `wait_next_event` nous permet entre autres choses de détecter que le bouton de la souris (cette petite bibliothèque graphique ne différencie pas les boutons de la souris) a été enfoncé ou au contraire qu'on l'a relâché.

À titre d'exemple, on peut détecter un clic de souris de la façon suivante :

```
let wait_click () =
  wait_next_event [Button_down]; wait_next_event [Button_up];;
```

Le résultat de cette fonction est alors de type `status`, qui comporte les informations `mouse_x` et `mouse_y`, c'est-à-dire les coordonnées graphiques de la souris au moment du clic.

On exploite alors la position de la souris de la façon suivante :

```
let s = wait_click () in
let x = s.mouse_x and y = s.mouse_y in
...
```

Le clavier

Il faut aussi lire des caractères au clavier, puisque le joueur peut taper « `q` » à tout moment d'une part et d'autre part on lui demande ses intentions quand le jeu est terminé.

Cette lecture peut se faire :

- soit en utilisant à nouveau la fonction `wait_next_event`, mais pour se qui est de la détection en cours de jeu, cela oblige à chaque évènement à vérifier s'il s'agit d'un clic ou d'une touche frappée au clavier ;
- soit en utilisant les fonctions spécialisées `key_pressed` détecte si une touche a été frappée et `read_key` qui lit ensuite un caractère au clavier.

2.6 Travail à réaliser

Il vous est demandé d'implémenter cette version simple le plus proprement possible. En particulier les paramètres du jeu devront pouvoir être aisément modifiés. Vous pourrez ensuite enrichir votre version comme vous le souhaitez en l'indiquant dans votre rapport.

Vous remettrez un petit rapport en même temps que votre programme, mentionnant les difficultés rencontrées, les choix que vous avez faits ainsi qu'une description précise de la façon dont vous avez exploité le marquage de la sélection lors de la suppression.

Annexe : un petit canevas

Voici une proposition de structure pour votre programme. Il n'est pas du tout obligatoire de la suivre, elle n'est indiquée que pour vous aider si vous rencontrez des difficultés à concevoir la structure de votre programme.

- `set_couleur` et `set_marque` qui servent à modifier la couleur ou la marque d'une case ;
- `frame`, `pattern` et `draw_board` qui servent à dessiner le tableau de jeu vide, une pièce et le tableau de jeu plein ;
- `closest_piece` qui détermine les coordonnées de la pièce la plus proche de l'endroit où l'on a cliqué ;
- `mark_connected_pieces` qui effectue le marquage ;
- `list_connected_pieces` qui transforme l'information de marquage de façon à ce qu'elle soit facilement exploitable pour l'opération de suppression ;
- `clear_columns` qui nettoie les colonnes sans se soucier de leur disparition éventuelle ;
- `empty_column` qui détecte si une colonne est vide, `suppress_column` qui supprime une colonne sans se poser de question, `suppress_columns` qui supprime les colonnes qui doivent l'être ;
- `update` qui met à jour la matrice et réaffiche le tableau de jeu ;
- `wait_click` qui détecte un clic souris ;
- `main_treat_events` qui gère tout ce qui suit un clic souris ;
- `new_game` qui initialise la matrice et dessine le tableau ;
- `end_in_game` qui détecte la frappe de la touche « `q` » au cours du jeu ;
- `empty_game` qui détecte si le tableau de jeu est vide ;
- `ask` qui demande ses intentions au joueur à la fin d'une partie ;
- `main` qui rassemble tout.