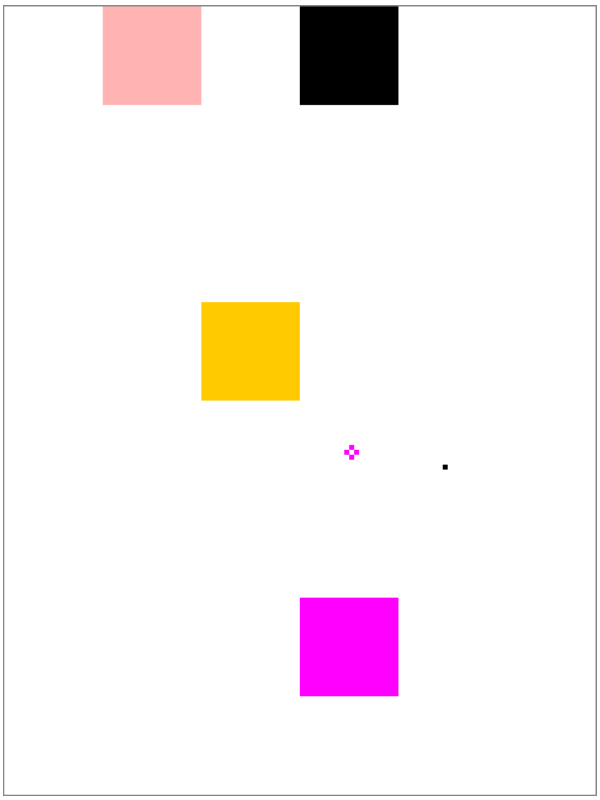
MESNIER Adrien
TS5

Jeu de casse-brique



Membres du groupe : MESNIER Adrien et BRETON Gwendal

Table des matières

Présentation du projet

Principe du jeu

Le programme est un jeu de casse brique assez simple :

On lance une balle pour casser des briques en cliquant à l'endroit où l'on veut qu'elle aille. Les briques descendent au fur et à mesure du jeu et il faut en casser le plus possible avant que les briques ne touchent le bas.
Nous nous sommes inspiré du jeu BBGEO qui existe sur smartphone.

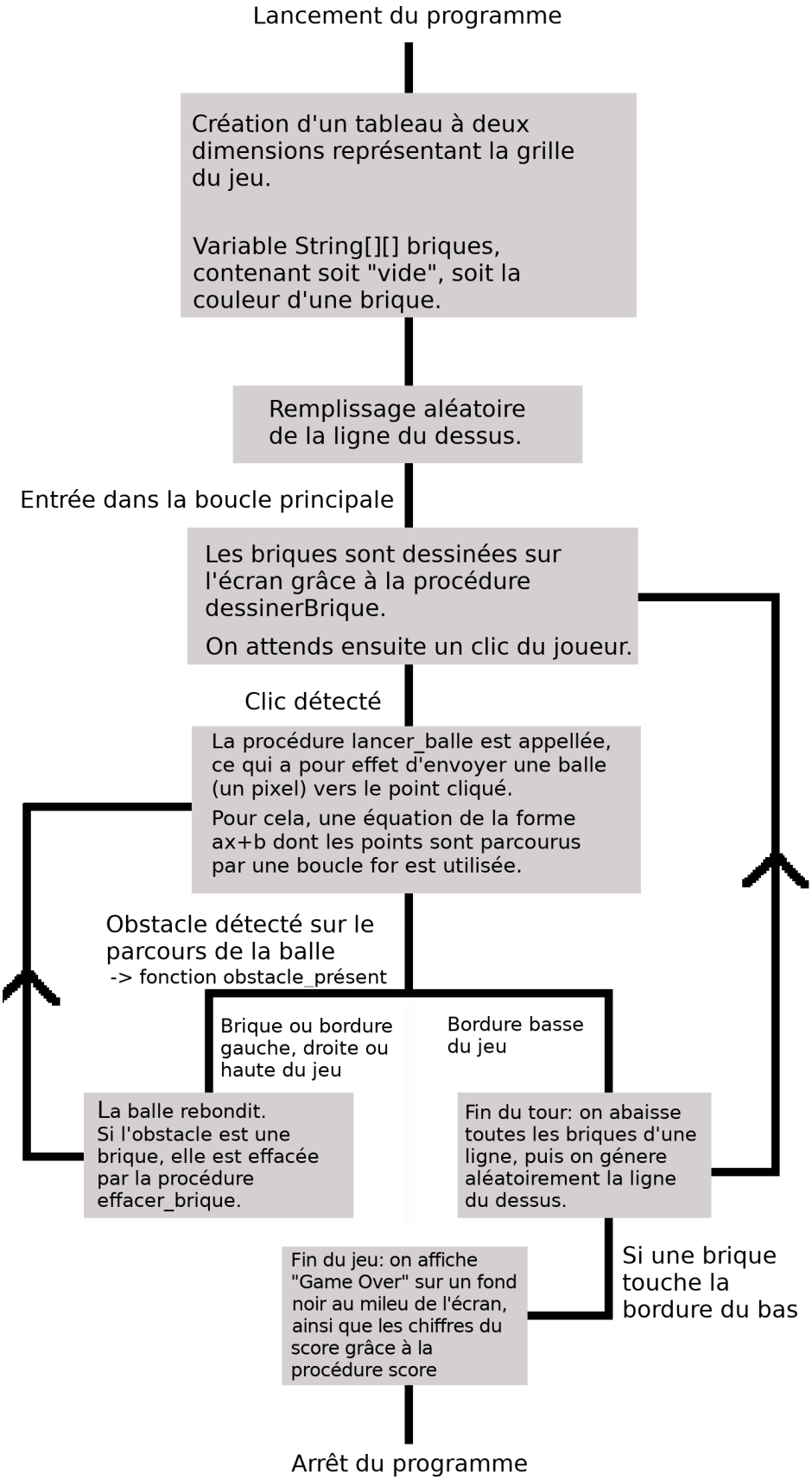
Objectifs du programme

L'objectif du programme est de générer une grille de jeu contenant des briques placées aléatoirement. Le joueur peut alors cliquer n'importe où sur la grille, ce qui entraîne un déplacement d'une balle représentée par un pixel, en direction du point cliqué.
Si le pixel rencontre une brique ou une bordure de la grille de jeu, il rebondit et la brique est effacée.
À la fin de chaque tour, toutes les lignes de briques sont abaissées et une ligne de briques aléatoires est générée au dessus.
Lorsqu'une brique touche le bas de la grille de jeu, la partie s'arrête. On affiche alors « Game Over » sur un fond noir, ainsi que le score, qui correspond au nombre de briques détruites.

Organisation du travail

Nous réalisions chacun une petite partie du programme puis nous mettions en commun lorsque nous avions fini.
Si une partie était plus difficile ou ne fonctionnait pas nous la réalisions ensemble.
Lorsque nous rencontrions des difficultés, nous recherchions sur Internet (openclassroom, ou nous demandions à notre professeur.

Fonctionnement global du programme



Détail du programme

Dans cette partie, je vais détailler le fonctionnement de certaines parties importantes du programme.

La procédure *lancer_balle*

La procédure lancer_balle est sans doute l'une des plus importantes et complexes du programme. Son but est de calculer la trajectoire en ligne droite d'une balle, puis de la faire suivre cette trajectoire à l'aide d'une boucle *for*.

Pour cela, selon ces cas d'utilisation, elle nécessite différents paramètres .

● Dans le cas où l'on veut envoyer la balle vers un point cliqué par le joueur :
La procédure prend en paramètre les coordonnées (x et y) du point de départ et du point d'arrivée. À partir de ces deux points, on peut calculer le a l'équation ax+b :

```
a = ( (float) yArrivee-(float) yDepart) / ((float) xArrivee - (float)xDepart);
```

Le a est calculé avec la formule a = $\frac{y_B - y_A}{x_B - x_A}$ Les entier représentant les coordonnées sont convertis en *float* avant le calcul afin de permettre plus de précision.
La direction dans laquelle doit aller la balle est calculée comme ceci :

```
 //si la balle part vers la gauche
if (xArrivee <= xDepart) direction = -1;
//Ou vers la droite
else if (xArrivee > xDepart) direction = 1;
```

Cette valeur de direction sera ensuite utilisée dans la boucle for pour incrémenter x : si direction vaut 1, x augmentera de 1 à chaque tour de la boucle et la balle ira donc vers la droite, tandis que si direction vaut -1, x diminuera de 1 et la balle ira vers la gauche.

● Dans le cas où la balle rebondit sur une brique ou une bordure :

Dans ce cas là, c'est plus simple : en effet pour obtenir l'équation de la trajectoire après le rebond, on part de l'équation de la trajectoire avant le rebond, et on inverse le *a* et/ou la direction selon le type de rebond.

C'est pour cela que la procédure lancer_balle peut accepter directement en paramètres une valeur de *a* et une direction, au lieu des coordonnées du point d'arrivé.

Dans tous les cas, la fonction calcule ensuite *b* avec la formule b = $y_A - a \times x_A$

La procédure, grâce à une boucle for, parcourt ensuite toutes les valeurs de x à partir du x du point de départ, dans la direction calculée précédemment. Pour toutes ces valeurs de x, elle calcule la valeur de y. À partir de ces deux valeurs, on sait où l'on se situe sur la grille de jeu. La procédure commence alors par vérifier qu'il n'y a pas d'obstacle à cet endroit grâce à la fonction obstacle_present.

-Si aucun obstacle n'est présent, la balle est dessinée avec la procédure dessiner_balle.

-Si l'obstacle est la bordure basse, on arrête la boucle et la procédure est donc terminée.

-Si l'obstacle est une brique, on calcule les coordonnées de la brique dans le tableau représentant la grille du jeu (car la taille du tableau en nombre de cases est différente de la taille de l'image en nombre de pixels). On appelle la fonction effacer_brique en lui donnant en paramètre ces coordonnées, ce qui aura pour effet de la faire disparaître de l'image et de définir sa valeur sur « vide » dans le tableau.

-Enfin, selon le type d'obstacle (horizontal, vertical ou coin), la procédure lancer_balle s'appelle elle-même (récurrence) en donnant en paramètre le point de l'impact avec l'obstacle, et l'inverse des variables *a* et/ou *direction :*

```
if(obstacle == "bordure_basse") {
     break;
}
else if(obstacle.indexOf("coin") != -1){
     lancer_balle(x, y, null, null, a, -direction);
}
else if (obstacle.indexOf("haut") != -1 || obstacle.indexOf("bas") != -1){
     lancer_balle(x, y, null, null, -a, direction);
}
else if (obstacle.indexOf("droit") != -1 || obstacle.indexOf("gauche") != -1){
     lancer_balle(x, y, null, null, -a, -direction);
}
```

La fonction *obstacle_present*

Le but de cette fonction est de déterminer si un obstacle est présent à un point donné de l'image du jeu. En paramètre, elle nécessite seulement les coordonnées x et y du point à vérifier.

La détection de l'obstacle se fait en plusieurs étapes :

-On vérifie d'abord que les coordonnées du point ne sont pas inférieures à 0, ou supérieures à la taille de l'image, ce qui signifierait que la balle est sorti de l'image du jeu. Dans ce cas, on retourne « bordure_basse », « bordure_haute », « bordure_droite »ou bien « bordure_gauche » selon la bordure dépassée.

-Si la balle est bien à l'intérieur de l'image du jeu, on vérifie qu'elle n'est pas sur une brique. Si c'est le cas, on retourne « aucun ».
Sinon, on détermine quel côté de la brique a été touché par la balle en calculant la coordonnée x des côtés droit et gauche de la brique, et la coordonnée y des côtés bas et gauche, puis en comparant avec les coordonnées x et y du point donné.
On peut ainsi retourner soit « brique_coin », soit « brique_droit » ou autre côté.

-Enfin, si l'on a détecté que le point se trouvait sur une brique mais qu'il est impossible de déterminer le coin ou le côté touché par la balle, la fonction retourne « bizarre ». C'est malheureusement assez souvent le cas et c'est pourquoi la fonction *obstacle_present* est à l'origine de certains bugs du programme (les balles qui ne rebondissent pas sur les briques).

Ce bug est dû au fait que la balle, si elle passe par toutes les valeurs de x sur un intervalle, peut en revanche sauter certaines valeurs de y (pour les trajectoires les plus verticales). Au moment où *obstacle_present* est appelée, la balle est donc déjà au milieu d'une brique et il est impossible de déterminer par quel côté de la brique elle est arrivée. Une solution possible mais que je n'ai pas eu le temps de mettre en place serait de parcourir toutes les valeurs de y, même si l'on ne dessine pas la balle à chaque fois.

La fonction *finDeTour*

La fonction *finDeTour* est appelée, comme son nom l'indique, à la fin de chaque tour. Son but est d'abaisser les briques et de déterminer si le joueur à perdu ou non.

Pour cela, deux boucles for imbriquées sont utilisées pour parcourir chaque case du tableau à deux dimensions *briques* contenant la grille du jeu, en partant de la ligne du bas.

```
for (int i = briques.length -1; i >= 0; i --) {
    for (int j = 0; j < briques[0].length; j ++) {
```

Ensuite, selon la ligne, différentes actions sont réalisées :

-Si l'on est sur la ligne du bas et qu'une case contient une valeur différente de « vide », cela signifie qu'une brique a touché la bordure du bas et donc que le joueur a perdu. La fonction retourne donc le boolean « false ».

-Si l'on est ni sur la ligne du bas, ni sur la ligne du haut, on remplace le contenu de la case par le contenu de la case de la même colonne sur la ligne au dessus : ainsi, les briques s'abaissent.

-Enfin, si l'on est sur la ligne du haut, il n'y a aucune brique au dessus à copier. On assigne donc pour chaque case la valeur retournée par la fonction brique_au_hasard (40 % de chance que ce soit une couleur aléatoire, 60 % de chance que ce soit « vide »).

Pour finir, la fonction retourne « true » pour signifier que la partie continue !

Bilan

À propos du programme

Le résultat auquel nous sommes arrivés, bien que loin d'être parfait, nous semble déjà complet et agréable à utiliser. Il respecte les objectifs que nous nous étions donnés au début du projet (réaliser un jeu de casse brique tout en gardant un code lisible et compréhensible). Pour cela, nous avons du faire des choix, notamment celui d'utiliser la proglet codagePixel plutôt qu'une solution d'affichage plus sophistiquées mais malheureusement trop complexe.

Le programme est loin d'être terminé : il reste des bugs importants que nous n'avons pas eu le temps de corriger, et des fonctionnalités que nous aurions aimé ajouter (par exemple un menu, des formes de briques différentes…).

À propos de la réalisation du projet

La réalisation de ce projet a été très enrichissante pour moi : en effet, malgré une expérience assez importante en programmation, je n'avais jamais développé de jeu comme celui-ci. L'utilisation d'équation pour représenter une trajectoire, ou encore le calcul de rebond, m'ont ainsi beaucoup interessé.

Le travail en équipe avec Gwendal, qui a lui aussi de bonnes compétences en programmation, a mené à des échanges intéressants sur différentes manières d'implémenter certaines fonctionnalités.

Finalement, je suis fier du programme que nous avons réalisés, et je suis persuadé que la réalisation de ce projet m'a apporté une plus grande ouverture d'esprit sur la programmation en général.