Final Project — Push and Clear

林秉軒、王 勻

January 18, 2022

1 遊戲規則介紹

在 6×6 的盤面中，玩家（實心圓圈）可以透過上下左右移動將整排箱子一次推到底，若六個箱子連成一直排或一橫排即可消除，在原位置留下點數（實心圓點），只要玩家或箱子通過放有點數的格子即可將點數收集成分數。每次玩家操作後都會隨機生成一個新的箱子，其中有一定的機率會產生不可移動的固定箱子(打叉的方格)，而當玩家無法移動時，遊戲便會結束。

2 設計邏輯

以陣列維護當前的盤面狀態，每次移動時便可快速計算下一個盤面的狀態以及分數，最後再隨機插入新的箱子即可。

3 程式碼

3.1 亂數模型（Random.jack）

我們採用線性同餘法作為亂數模型。在初始化 `init()` 後，`gen()` 會產生一個介於 $0 \sim 172$ 的亂數，而 `next(k)` 則會產生一個介於 $0 \sim k-1$ 的亂數（藉由過濾不小於 $\lfloor \frac{mod}{k} \rfloor \times k$ 的亂數使機率分布均等）。

```
class Random {
    static int K, mod;
    function void init() {
        let K = 49;
        let mod = 173;
        return;
    }
    function int gen() {
        let K = (K * 31);
        let K = K - (K / 173 * 173);
        return K;
    }
    function int next(int range) {
        var int up, res;
        let up = mod / range * range;
        let res = up;
        while (~(res < up)) {
            let res = Random.gen();
        }
        let res = res - (res / range * range);
        return res;
    }
}
```

3.2 遊戲模型（Game.jack）

3.2.1 fields, constructor

儲存並初始化物件在螢幕上的大小排版，盤面狀況、玩家位置、總分和盤面狀態編碼。

```
field int SQUARE_WIDTH, GAP_WIDTH, BOARD_WIDTH;
field int OFFSET_X, OFFSET_Y;

field int N;
field Array board;
```

```
field int step;
field int playerX;
field int playerY;
field int totalPoint;

static int EMPTY_CELL, PLAYER, SP_ITEM, POINT, NORMAL_BLOCK,
   STABLE_BLOCK;
// 0 - empty cell
// 1 - player
// 2 - a special item
// 3 - a point
// 4 - a normal block
// 5 - a stable block
constructor Game new(int N_) {
    let N = N_;
    let board = Array.new(N * N);
    let SQUARE_WIDTH = 30;
    let GAP_WIDTH = 5;
    let BOARD_WIDTH = (N * SQUARE_WIDTH) + ((N + 1) * GAP_WIDTH)
       ;
    let OFFSET_X = (512 / 2) - (BOARD_WIDTH / 2);
    let OFFSET_Y = (256 / 2) - (BOARD_WIDTH / 2);

    let EMPTY_CELL = 0;
    let PLAYER = 1;
    let SP_ITEM = 2;
    let POINT = 3;
    let NORMAL_BLOCK = 4;
    let STABLE_BLOCK = 5;

    return this;
}
```

3.2.2 Basic methods: `ended`, `isInBoard`, `setBoard`

`ended()`：透過 `move(..., false)` 判斷是否四個方位皆不可移動。
`isInBoard(x, y)`：判斷 (x, y) 座標是否在界內。
`setBoard(id, x, y, type)`：將 (x, y) 座標的狀態更改為 `type`，並更新此格的畫面顯示。

```
method boolean ended() {
    return ~(move(-1, 0, false) | move(1, 0, false) | move(0,
       -1, false) | move(0, 1, false));
}

method boolean isInBoard(int x, int y) {
    return (~(x < 0)) & (x < N) & (~(y < 0)) & (y < N);
}

method void setBoard(int id, int x, int y, int type) {
    let board[id] = type;
    let x = (x * (SQUARE_WIDTH + GAP_WIDTH)) + GAP_WIDTH;
    let y = (y * (SQUARE_WIDTH + GAP_WIDTH)) + GAP_WIDTH;
    do Game.drawCell(type, x + OFFSET_X, y + OFFSET_Y,
       SQUARE_WIDTH);
    return;
}
```

3.2.3 `move(dx, dy, apply)`

判斷能否往 (dx, dy) 方向移動，若 `apply = true`，則同時更新盤面。

流程

1. 計算玩家位置到牆壁（或固定箱子）之間的點數、箱子數和空格數。
2. 藉由點數、空格數總和是否為 0 判斷此方向是否有空位移動。
   若 `apply = false`，可直接回傳。
3. 將點數累計入分數、箱子靠牆排列，並更新玩家位置。

4. 呼叫 elimination 函數消除完整的行列，並更新移動步數與分數畫面顯示。

```
method boolean move(int dx, int dy, boolean apply) {
    var int x, y, id;
    var int tx, ty;
    var int empty, point, sp_item, normal_block;
    var boolean found, moved;
    let x = playerX + dx;
    let y = playerY + dy;
    let empty = 0;
    let point = 0;
    let sp_item = 0;
    let normal_block = 0;
    let found = false;
    while ((~found) & isInBoard(x, y)) {
        let id = (x * N) + y;
        if (board[id] = STABLE_BLOCK) {
            let found = true;
        } else {
            if (board[id] = EMPTY_CELL) {
                let empty = empty + 1;
            }
            if (board[id] = POINT) {
                let point = point + 1;
            }
            if (board[id] = SP_ITEM) {
                let sp_item = sp_item + 1;
            }
            if (board[id] = NORMAL_BLOCK) {
                let normal_block = normal_block + 1;
            }
            let x = x + dx;
            let y = y + dy;
        }
    }
    if (~apply) {
        return ~((empty + point + sp_item) = 0);
    }
    // board[x, y] now is out of map or is a stable block
```

```
    let tx = x;
    let ty = y;
    let x = tx;
    let y = ty;
    while (~((x = playerX) & (y = playerY))) {
        let x = x - dx;
        let y = y - dy;
        let id = (x * N) + y;
        do setBoard(id, x, y, EMPTY_CELL);
    }
    let x = tx;
    let y = ty;
    while (normal_block > 0) {
        let x = x - dx;
        let y = y - dy;
        let id = (x * N) + y;
        do setBoard(id, x, y, NORMAL_BLOCK);
        let normal_block = normal_block - 1;
    }
    let x = x - dx;
    let y = y - dy;
    let id = (x * N) + y;
    do setBoard(id, x, y, PLAYER);
    let moved = ~((playerX = x) & (playerY = y));
    let playerX = x;
    let playerY = y;

    let totalPoint = totalPoint + point;

    do elimination(true);
    do Output.moveCursor(0, 0);
    do Output.printInt(totalPoint);

    if (moved) {
        let step = step + 1;
    }
    return moved;
}
```

3.2.4 elimination(apply)

將所有完整的行列消除，並回傳是否有任一行列被消除。

流程

1. 枚舉所有直行與橫列，並將可消除的行列位置打上標記。
2. 若 apply = false，可根據是否有位置被標記判斷有無消除，並直接回傳。
3. 顯示消除提示文字，並將所有被標記的位置狀態設為點數

```
method boolean elimination(bool apply) {
    var Array bingo;
    var int i, j, id;
    var boolean flag, hasBingo;
    let bingo = Array.new(N * N);
    let hasBingo = false;
    let id = 0;
    while (id < (N * N)) {
        let bingo[id] = false;
        let id = id + 1;
    }
    let i = 0;
    while (i < N) {
        // vertical
        let j = 0;
        let flag = true;
        while (j < N) {
            let id = (i * N) + j;
            if (~((board[id] = NORMAL_BLOCK) | (board[id] =
               STABLE_BLOCK))) {
                let flag = false;
            }
            let j = j + 1;
        }
        if (flag) {
            let j = 0;
```

```
            while (j < N) {
                let id = (i * N) + j;
                let bingo[id] = true;
                let hasBingo = true;
                let j = j + 1;
            }
        }
        // horizontal
        let j = 0;
        let flag = true;
        while (j < N) {
            let id = (j * N) + i;
            if (~((board[id] = NORMAL_BLOCK) | (board[id] =
               STABLE_BLOCK))) {
                let flag = false;
            }
            let j = j + 1;
        }
        if (flag) {
            let j = 0;
            while (j < N) {
                let id = (j * N) + i;
                let bingo[id] = true;
                let hasBingo = true;
                let j = j + 1;
            }
        }
        let i = i + 1;
    }
    if (~apply) {
        return hasBingo;
    }
    if (hasBingo) {
        do Output.moveCursor(0, 25);
        do Output.printString("line clear!");
        do Sys.wait(500);
        do Output.moveCursor(0, 25);
        do Output.printString("           ");
```

```
    }
    let id = 0;
    while (id < (N * N)) {
        if (bingo[id]) {
            let i = id / N;
            let j = id - (i * N);
            if (board[id] = NORMAL_BLOCK) {
                do setBoard(id, i, j, POINT);
            } else {
                do setBoard(id, i, j, POINT);
                // let board[id] = SP_ITEM;
            }
        }
        let id = id + 1;
    }
    do bingo.dispose();
    return hasBingo;
}
```

3.2.5 generateNewBlock()

隨機在空白位置生成新箱子，且有 $\frac{10\sqrt{steps}}{3}$ 的機率是固定箱子。為降低自動生成馬上造成消除的可能，生成箱子的位置一旦造成消除，便會重新選擇一個，至多重複三次。而為了避免自動生成固定箱子馬上困住玩家導致遊戲結束，若在選定的位置放置固定箱子後會導致遊戲結束，便以一般的箱子取代之。

```
method void generateNewBlock() {
    var int pos, x, y;
    var int iter, org;
    var boolean flag;
    let iter = 0;
    let flag = true;
    while ((iter < 3) & flag) {
        // try 3 times, if no elimination, break.
        let pos = Random.next(N * N);
        while (~((board[pos] = EMPTY_CELL) | (board[pos] = POINT
           ))) {
```

```
            let pos = Random.next(N * N);
        }
        let org = board[pos];
        let x = pos / N;
        let y = pos - (x * N);
        let board[pos] = NORMAL_BLOCK;
        let flag = elimination(false);
        let iter = iter + 1;
        let board[pos] = org;
    }
    if (Random.next(100) < (Math.sqrt(step * 100) / 3)) {
        do setBoard(pos, x, y, STABLE_BLOCK);
        if (ended()) {
            do setBoard(pos, x, y, NORMAL_BLOCK);
        }
    } else {
        do setBoard(pos, x, y, NORMAL_BLOCK);
    }
    do elimination(true);
    return;
}
```

3.2.6 generateNewBlock()

初始化 fields，並隨機放置六個一般箱子與玩家位置。

```
method void init() {
    var int i, pos;
    let i = 0;
    while (i < (N * N)) {
        let board[i] = EMPTY_CELL;
        let i = i + 1;
    }
    let i = 0;
    while (i < N) {
        do generateNewBlock();
        let i = i + 1;
```

```
    }
    let pos = Random.next(N * N);
    while (~(board[pos] = EMPTY_CELL)) {
        let pos = Random.next(N * N);
    }
    let board[pos] = PLAYER;
    let playerX = pos / N;
    let playerY = pos - (playerX * N);
    let totalPoint = 0;
    let step = 0;
    return;
}
```

3.3 主函數（Main.jack）

初始化、主迴圈（輸入 → 移動 → 生成 → 顯示）

```
class Main {
    function int readOperation() {
        var int ch;
        let ch = Keyboard.readChar();
        do Output.backSpace();
        if ((ch = 130)) { return 0; }
        if ((ch = 131)) { return 1; }
        if ((ch = 132)) { return 2; }
        if ((ch = 133)) { return 3; }
        return -1;
    }
    function void main() {
        var Game game;
        var int direction, dx, dy;
        var boolean moved;

        let game = Game.new(6);

        do Random.init();
        do game.init();
```

```
        do game.draw();
        while (~game.ended()) {
            let direction = Main.readOperation();
            let moved = false;
            if (direction = 0) {
                let dx = -1;
                let dy = 0;
            }
            if (direction = 1) {
                let dx = 0;
                let dy = -1;
            }
            if (direction = 2) {
                let dx = 1;
                let dy = 0;
            }
            if (direction = 3) {
                let dx = 0;
                let dy = 1;
            }
            if (~(direction = -1)) {
                let moved = game.move(dx, dy, true);
            }
            if (moved) {
                do game.generateNewBlock();
            }
        }
        do game.draw();
        do Output.moveCursor(0, 25);
        do Output.printString("game over!");

        do game.dispose();
        return;
    }

}
```