

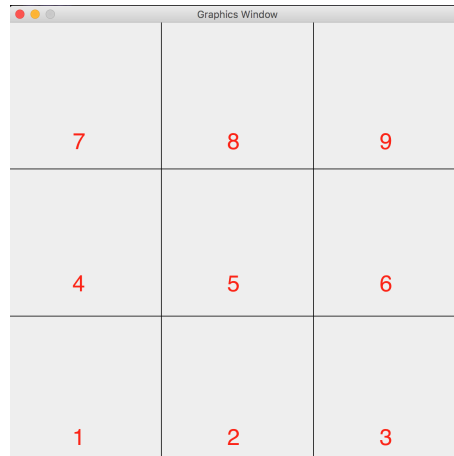
COM6471 Assignment 2: Herding Cats!

Dr Ramsay G Taylor

October 28, 2018

1 Outline

This assignment requires you to use Java to implement a simple game. The game is played on a 3 x 3 grid where the squares are numbered from 1 to 9 in the pattern of a standard computer keypad. This numbering is shown in Figure 1.



7	8	9
4	5	6
1	2	3

Figure 1: The grid numbering

The game involves 10 Cats and one Dog. The Cats and Dog are each located in one particular square on each turn of the game. The player controls the Dog, and the Cats move (mostly) in response to the Dog's position.

The game starts with the 10 Cats randomly distributed across the grid. On each turn the player enters a number and the Dog is moved to that square. The Cats should then move 1 square, according to the following rules:

- Each Cat picks a number between 1 and 6

- If the number is 1-3, then the cat moves *away* from the dog. This should be to the square directly away from the dog, unless this would take the Cat off the edge of the grid, in which case it should stay still.
- If the number is 4 or 5, the Cat should move *beside* the Dog — i.e. it should move to a square adjacent to its start square and adjacent to the Dog. For example, in the position shown in Figure 2, a Cat on square 5 who rolled a 4 or 5 could move to either square 8 or square 6.
- If the number is a 6 then the Cat should move to *the same* square as the Dog
- If the Dog is on the same square as the Cat at the start of the turn, then the cat can move randomly to any adjacent square.

One position in the game is shown in Figure 2, with Cats represented by the letter *C* and the Dog represented by the letter *D*. The player has chosen to move the Dog to square 9 in this example.

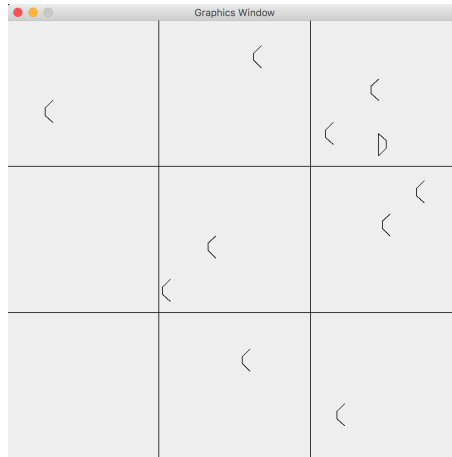


Figure 2: An example of the board

The object of the game is to herd the Cats into a single square. It doesn't matter which square, once all the Cats are in one square the player wins the game. (Note: this game is based on an English idiom about the impossibility of herding cats — its not intended to actually be likely that you can win!!)

The game could be expanded to include other rules, e.g.: the Dog could only be allowed to move to squares adjacent to its current square, or the board could be made larger.

2 Task

Your assignment is to implement this game in Java.

- You must create a `Cat` class that models the individual Cats. It must contain an instance variable that tracks which square the Cat is on, and a `void move(int dog)` method that takes the Dog's current square as an argument and updates the Cat's square appropriately.
- You should create other classes and methods as you choose to form a structured, Object-Oriented system.
- You must create a graphical display of the grid and the Cats and Dog. Just using letters, as in Figure 2 is perfectly fine, but you can create some more imaginative symbols if you want to. You are encouraged to use the `EasyGraphics` class from the `sheffield` package, but you may use standard Swing classes if you prefer. If you use esoteric packages from weird libraries then you may not receive marks for either code clarity, or for its function if its operation is not obvious, and the markers will not go and download stuff from the internet just to make your code run!
- The game should randomly place the 10 Cats and then prompt the user for a square number to place the Dog (you can use `EasyReader` for this and have the user enter the number in the terminal). The Cats should each have their `move(dogsquare)` method called to update their position. The grid should then be re-drawn and the user prompted for another square.
- The game should end if either all the Cats are in the same square, or the user enters a number greater than 9. It should print a message to say "You've won!" or "Exiting" to show which event cause it to end.
- If you `import java.util.Random` you can create an new `Random` object and use its `nextInt(6)` method to produce a number from 0 to 5 (i.e. from 0 to less than 6)

3 Submission and Marking

You should submit your `.java` files through MOLE. The MOLE submission page will be configured to accept multiple files and you should ensure that all the files you have written are included.

The marking scheme will be as follows:

70 — 100%	Classes, methods, and instance variables used extensively and appropriately. Clear code layout. Useful, descriptive comments. All of the described game functionality implemented, and some additional functionality or interface improvements included
60 — 70	Classes, methods, and instance variables used appropriately. Clear code. Some meaningful comments. All of the described game functionality implemented.
50 — 60	Classes, methods, and instance variables used. Readable code, if not perfectly organised or presented. Most of the game functionality is present.
40 — 50	Some use of Classes, methods, and instance variables. Confusing but basically functional code. A recognisable proportion of game functionality present.
0 — 40	Limited or no OO features. Virtually no recognisable components of the required game present. Meaningless code fragments.

You are reminded that the University of Sheffield takes the use of unfair means very seriously. The code you present for this assignment must be entirely your own work. Code that is **plagiarised** — that is, copied from somewhere else — is not acceptable and you will receive zero for this assignment and be called to a departmental unfair means panel where it will be investigated further. If you are found to have **knowingly allowed your work to be plagiarised** you will also receive zero and be further investigated.

Since the purpose of these assignments is to test **your knowledge** of Object Oriented design and Java, you must also avoid **collusion** — that is, discussing your design or implementation with your classmates to such a degree that you end up submitting similar work. Even if you have produced the code independently, part of the object of the assignment is to evaluate your Object Oriented design approach, so you must develop your solution independently.