Linköping University
Department of Computer and Information Science (IDA)
The UPP group 2025-09-22

Pacman

Aim

In this laboration you will implement the game logic of the ghosts in the classic arcade game Pacman. You will practice reading and understanding parts of already written code, and extending it.

Scenario

You are working in a group tasked with implementing a version of the classic arcade game Pacman. Your task is to write code that represent three of the ghosts in the original. Your code will be integrated into an already existing code base, so it's important to take into account how the rest of the game is structured. You are also expected to write a test for your implementation.

Quality requirements

Your solution need to achieve the below points on correctness, usability, changeability, and integrability.

Correctness

Your colleagues should not be able to use your module in an incorrect way. It should only be possible to create complete and correct objects. Incorrect usage of your module should result in compilation errors whenever possible, and secondarily it should result in runtime errors. Your module should always generate complete and correct results. Your module should not generate immaterial or incorrect results. Errors stemming from using the classes should be reported to the calling code. The caller is responsible for filtering what is reported to the user. You will need to think about how your colleagues are supposed to catch errors.

Usability

The code for you module should follow standard file separation, i.e. an implementation file and a header file. It needs to be clear what code is supposed to be used by your colleagues, and what code they should ignore. Using code that they should ignore should result in compilation errors. Your module need to follow C++ conventions wherever possible. Compiling your module should not result in any warnings or errors.

Changeability

You should assume that your code will be used by your colleagues for its intended purpose, but they also want to reuse the code in other programs as well. To this end, you need to make sure that you can modify the internal representation of the ghosts without your colleagues having to modify their own code. Their code should not need to be modified or updated to different syntax or functionality

between the old and updated module. In other words, your colleagues should not have to update their code even if you change the underlying structure.
The program needs to be easy to expand. It should be possible to add new ghosts without having to modify existing code in your module. Your test program and your colleagues' code should only need to be extended with whatever is new, without modifying already existing code.

Integrability

You should apply code written by your colleagues to solve problems where applicable. The given code should work without modifications. Your code should follow the naming convention and code style that is already established in the given code.

Functional requirements

Your task is to implement the classes `Blinky`, `Pinky`, and `Clyde` that are a part of a polymorphic class hierarchy. The ghosts' specific behavior is described in the *Pacman in a nutshell* section further down.
Each class implements

- `get_chase_point()` returns the point the ghost is aiming towards during its "chase" state.
- `get_scatter_point()` returns the point the ghost is aiming for during its "scatter" phase.
- `get_color()` returns a string representing the ghost's color. E.g. `"red"`, `"green"`, or `"blue"`.
- `set_position(Point)` changes the ghost's current position to a different one on the game board.
- `get_position()` returns the ghost's current position.

On top of that, `Blinky` need to implement the following

- `is_angry()` returns true if Blinky is angry, false otherwise.
- `set_angry(bool)` sets the "angry" status of Blinky.

You will use object oriented principles to reuse code whenever possible. Duplicating code between the different classes is not allowed. See "Relevant concepts" for a list of concepts to study. You should start by completely reading these instructions, reading up on each of the concepts and studying the given code in `main.cc`, `given.h`, and `given.cc`. Your ghost classes should be declared and defined in their own `.h` and `.cc` file, with appropriate names. You can use one `.h` for all of the declarations, and one `.cc` file for the definitions, as long as you can think of a reasonable motivation for it.

Testing

Your solution need to be tested through a terminal interface described below. The test program visualizes the objects placement and gives the tester the option to change different properties. The test program should be implemented as a class, with clearly defined member functions with a clear purpose. The test program needs to support the following:

- move a ghost by supplying its color and a new position
- move the player with the command `pos` and a new position
- set the player's direction with the command `dir` and the new direction
- change between "chase" and "scatter" with the commands `chase` and `scatter`
- terminate the program with `quit`
- make relevant ghosts angry with `anger`

Your test program should be written in such a way that there isn't any code duplication, while being as clear as possible.

Test requirements

Writing the test program without code duplication and making it as clear as possible is a great challenge. It's easy to get stuck in deep if-statements with a indentation depth that makes it harder to read. Your task is so avoid these problems.
Example 1 shows how the test program is expected to work. Upper case letters are used to mark a ghost's position and the corresponding lower case letter is used to mark that ghost's target. After each command the user enters the board is updated. The user's entry is marked with **bold** text.

Test output explained: Each position on the game board is drawn as two characters in width. Coordinate `(0,0)` is located on the bottom left corner. The game board below has the size `(7,7)`, so the top right corner has the coordinate `(6,6)`. The game board used here is shrunk to save space, but your implementation should of course be the full size (see WIDTH, HEIGHT in `given.cc`). A ghost is drawn as the first character in a position, and should represented as the first letter of its name in the ghost's color. The player is drawn as the second character of a position and should be represented with `@`.

Tip: To find the right letter to draw for each ghost you can fetch the first letter in that ghost's color and use the functions `toupper`/`tolower` to convert between upper and lower case. Note that you will need to use `static_cast` to convert the return value to a character.

Given code

In the given files you will find code to get you going with the assignment. The file `main.cc` contains the start of your test program. Extend it with your own code to fulfill the testing requirements above.

`given.h` and `given.cc` contains a selection of code from the complete game that you will need to use to create a working implementation. The `Pacman` class may not be modified. The `Point` struct may be extended with your own operators if you feel the need. The files need to be modified so the program compiles correctly.

`Point` is used to both represent positions on the game board, and the player's direction. In the case of the `Point` being used for direction, the following values will be used: `(1,0), (0,1), (-1,0), (0,-1)`, where the numbers represent positive or negative direction in the x- and y-direction.

Example 1

```
$ ./a.out
+--------------+
|              |
|  R           |
|      P       |
|              |
|        O     |
|  r@  p       |
|o             |
+--------------+
> red 2 7
+--------------+
|    R         |
|              |
|      P       |
|              |
|        O     |
|  r@  p       |
|o             |
+--------------+
> pos 4 6
+--------------+
|    R   r@  p |
|              |
|      P       |
|              |
|        O     |
|              |
|o             |
+--------------+
> scatter
+--------------+
|p   R    @  r |
|              |
|      P       |
|              |
|        O     |
|              |
|o             |
+--------------+
> quit
```

Linköping University
Department of Computer and Information Science (IDA)
The UPP group 2025-09-22

Assignment

You need to show that you have understood how the relevant concepts in C++ are used and work by implement the program module such that all requirements are fulfilled. Relevant concepts are listed below.

Relevant concepts

The following concepts are part of this assignment. Concepts not included are those that you should have grasped in earlier assignments or courses.

- Object-orientation
- Object-oriented analysis
- Object-oriented design
- Classes
  - Inheritance
  - Polymorphism
  - Reference member variables (`&`)
  - Constant member variables (`const`)
  - Constant member functions (`const`)
  - (Static member variables (`static`))
- File separation
  - Declaration file
  - Implementation file
  - Include guards

Linköping University
Department of Computer and Information Science (IDA)
The UPP group 2025-09-22

Graphical Pacman

When your implementation and test program are working correctly you might want to actually try out your ghosts in a "real" game of Pacman. Luckily, your colleagues finished the rest of the game and uploaded it for you as the `full_game.zip` archive. You should download it and integrate your code according to the instructions and play a game or three.

Note: You might need to modify function- and parameter names in your code if you have deviated from above descriptions.

Pacman in a nutshell

In the classic arcade game Pacman the player is supposed to pick up all pieces of food without being caught by the three ghosts (usually 4) that are hunting Pacman. The important parts for the assignment is how the ghosts are moving around. A ghost picks its target position in relation to where the player character is, and plans a route based on that. The rules for how a ghost picks a target is different per ghost, and what the state the game currently is in.

The ghosts have two states, "chase" and "scatter". During the "chase" state the ghosts chase the player character. This means that they are picking their target based on where Pacman is, and plans a route to get there. During "scatter" the ghosts flee towards a predetermined corner of the map. The game switches between these states with predefined period. You will find a description of each ghosts' behavior in the table below.

	Color	**"chase"-target**	**"scatter"-target**
Blinky	Red	Pacman's position	Top right corner (or "chase"-target if angry)
Pinky	Pink	Two steps in front of Pacman, based on Pacman's current direction	Top left corner
Clyde	Orange	Pacman's position if Clyde is more than n steps away from Pacman. Otherwise its "scatter"-target.	Bottom left corner

Note: You are only meant to implement code that calculate what position a ghost should target, not what route to take.

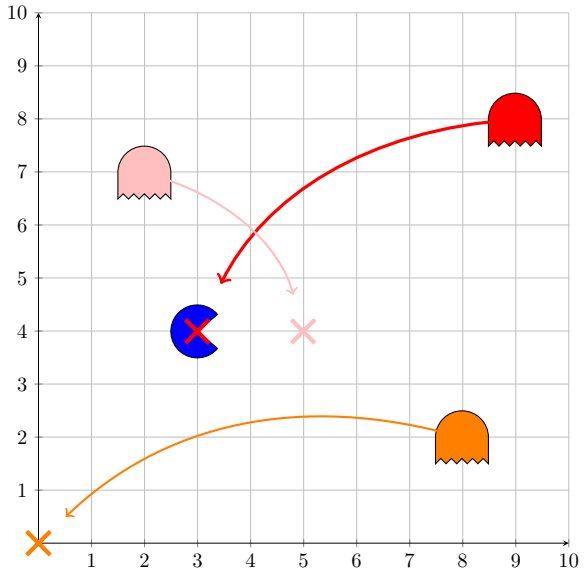

Figure 1: Pacman's position is (3,4), and its direction is to the right (1,0). The ghosts are currently in "chase". Therefor Blinky is targeting (3,4). Pinky's target is two steps in front of Pacman, (5,4). The distance between Pacman and Clyde is less than n (in this case 6) so its target is (0,0).

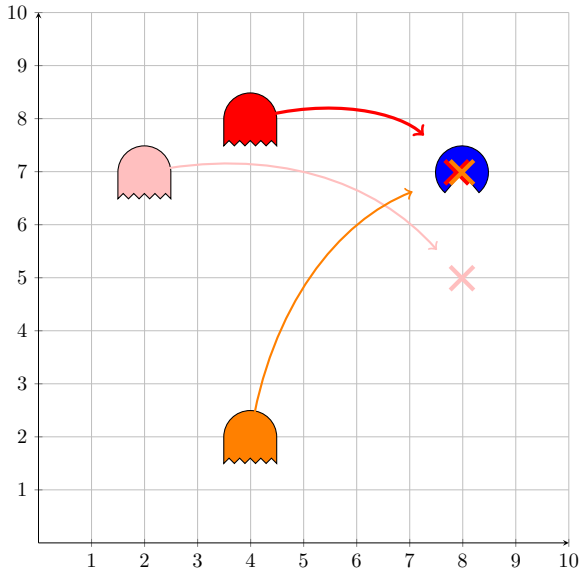

Figure 2: Pacman's position is (8,7) and directed downwards (0, -1). The ghosts are in "chase". Blinky's target is (8,7). Pinky is targeting two steps in front of Pacman, (8,5). The distance between Pacman and Clyde is greater than n (which in this case is 6) and target (8,7).