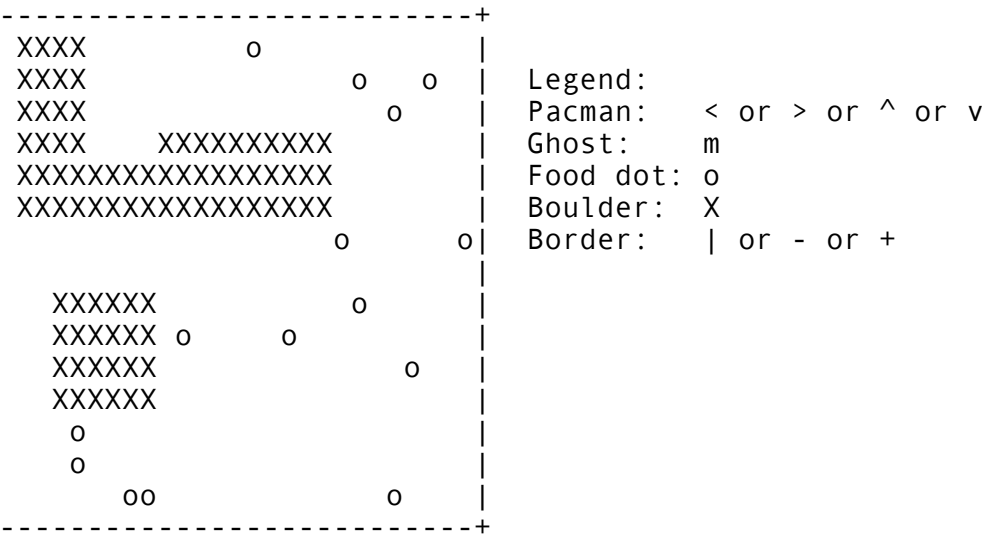
COMP 11 Spring 2016 Project 1: Pacman

Phase 0: Due March 29, 2016, 11:59 PM
Phase 1: Due April 1, 2016, 11:59 PM
Phase 2: Due April 4, 2016, 11:59 PM
Phase 3: Due April 8, 2016, 11:59 PM

Introduction

The second project of the semester is a version of the classic video game Pacman, in which the player (Pacman) charges around a board eating food dots and avoiding ghosts. In your version you will have to run around walls to find dots on the screen, and the ghosts will chase you with a rudimentary AI (Artificial Intelligence). Here is an example board from the game:



Notice that there are similarities to Snake -- you should be able to re-use some of your code between Snake and Pacman. Note also that while our Pacman looks like a snake head, it is backwards. “<” means that Pacman is moving to the right, with its mouth open (and “v” is up, etc.) (Wakawaka)

This project will rely heavily on the use of objects in the form of C++ classes. Each class should have its own .cpp file and .h (header) file. We have provided header files for the classes that you must use, and we have provided two .cpp files. You will have to write the remaining .cpp files.

Getting Started

Run `/comp/11/files/pacman/pacman11` to play the game.
(You can run `/comp/11/files/pacman/pacman11 slow` to play the game in stop motion).

Run `/comp/11/files/pacman/getfiles`
This will give you 9 files.

Files you are given	
`main.cpp`	This holds the `main()` function and **cannot be changed**.
`constants.h`	All of your constants will go in this file. We give you a lot, but feel free to add more if you extend the functionality of the game.
`game.h`	Header file for the Game class, which is the main Pacman game
`game.cpp`	Empty Game class that just contains the `print_manual()` function.
`pacman.h`	Header file for the Pacman class, which is a Pacman (the player)
`ghost.h`	Header file for the Ghost class
`dot.h`	Header file for the Dot class. These are the pellets that Pacman eats.
`boulder.h`	Header file for the Boulder class. These are the blocks in the board that Pacman cannot go through.
`boulder.cpp`	The full implementation of the Boulder class.
`termfuncs.h`	The header file for termfuncs, which includes functions like `screen_clear()`.
`termfuncs.cpp`	The full implementation of termfuncs.
`Makefile`	This can be used to compile the game at the end.

Files you will write	
`game.cpp`	The full implementation of the Game class.
`pacman.cpp`	The full implementation of the Pacman class.
`ghost.cpp`	The full implementation of the Ghost class.
`dot.cpp`	The full implementation of the Dot class.

Do not be scared of this many files! Each file has its own class, and most classes are going to be short. The largest file will most likely be `game.cpp`, where you will actually play the game.

To compile: `clang++ -Wall -Wextra -g *.cpp -o pacman`

Rules of the Game

- Pacman moves around the board with the following controls. Unlike Snake, there is *just* moving. There is no need to rotate and then move forward.

- Pacman starts in the center of the board, and he is facing up (v).
- Pacman can go through walls.
- Pacman can move into any spot *except* a spot occupied by a boulder.

- Boulders are placed randomly on the board, but they can not overlap the center spot in the board because this is is where Pacman starts.
- On level *x*, there are *x* boulders on the board (1 boulder on level 1, 2 on level 2, etc.).

- Dots are placed randomly on the board, but they cannot overlap the center spot in the board or any of the boulders.
- On level *x*, there are *5x* dots at the beginning of the level (5 dots on level 1, 50 on level 10).

- Ghosts begin in the four corners on the board.
- If Pacman moves into a ghost, or a ghost moves into Pacman, the game is over and the player loses.
- Every time a ghost moves, it moves towards Pacman.
- Ghosts cannot wrap around the board, but they *can* move through boulders.

- There are 10 levels.
- If Pacman successfully eats all of dots in one level, it advances to the next level.
- At the start of each level, the ghosts are in the corners and Pacman are in the center.
- If Pacman beats level 10, the game is over and the player wins.

- 1 point will be awarded for each dot eaten.
- 10 points will be awarded for each level completed.
- Score will be displayed at the bottom of the game.

Program Structure

You cannot remove any of the public functions that we describe in the header files, and each public function must work as described.

Controls	
a	Face head left and move one spot to the left
s	Face head down and move one spot to down
d	Face head right and move one spot to the right
w	Face head up and move one spot to the up

Boulder Class

This class is provided for you and represents the boulders onto the board.

- The top-left corner coordinates of a boulder are randomly generated
- The height and width of a boulder is constant
- A boulder can printed on the board

Dot Class

This class is provided for you and represents the dots onto the board.

- The coordinates of a dot is randomly generated, but it cannot overlap boulders
- A dot can printed on the board

Pac-Man Class

This class is responsible for setting the location of the Pac-Man character, moving and placing it on the board, and determining if it is alive or not. All of the moving logic happens inside the class. This includes going through walls and *not* going through boulders.

Ghost Class

This class is responsible for placing a ghost on the board and moving it closer to the Pac-Man.

Game Class

This class contains all of the objects that game needs (Pacman, Ghosts, Dots, Boulders) and places them on the board. This class also runs the gameplay.

Class Interfaces

Below is a list of the public interface and included private members for each class. You cannot remove anything from the public interface, but you can add to it. Additionally, you can (and should) add to the private members where necessary.

	Public Interface	Included Private Members

Game	`Game();`
	
	`void print_manual();`
	`void run();`
		`char board[ROWS][COLS];`
		
		`Pacman p;`
		`Ghost ghosts[NUM_GHOSTS];`
		`Dot dots[NUM_DOTS];`
		`Boulder boulders[NUM_BOULDERS];`

Pacman	`Pacman();`
	
	`void center();`
	`bool move(char[ROWS][COLS], char command);`
	`void place_on_board(char[ROWS][COLS]);`
	`int get_num_moves();`
	
	`void set_col(int);`
	`void set_row(int);`
	`int  get_col();`
	`int  get_row();`
	`bool is_at(int row, int col);`
	
	`void add_to_score(int n);`
	`int get_score();`
	
	`void die();`
	`bool is_alive();`
		`bool alive;`
		`char head;`
		`int row, col;`
		`int num_moves;`
		`int score;`

Ghost	`Ghost();`
	
	`void set_location(int row, int col);`
	`bool move(char[ROWS][COLS], int r, int c);`
	`void place_on_board(char[ROWS][COLS]);`
	`bool is_at(int row, int col);`
		`int row;`
		`int col;`

Boulder	`Boulder();`
	
	`void place_on_board(char[ROWS][COLS]);`
		`int top_left_r;`
		`int top_left_c;`
		`int height;`
		`int width;`
		
		`bool overlaps_middle();`

Dot	`Dot();`
	
	`void set_random_location(char[ROWS][COLS]);`
	`void place_on_board(char[ROWS][COLS]);`
	`bool is_at(int row, int col);`
	
	`void set_eaten(bool);`
	`bool was_eaten();`
		`int row;`
		`int col;`
		`bool eaten;`

Phase 0: Due March 29, 2016, 11:59 PM

Summary:

- Your program must print the board with 3 boulders and 15 dots

For Phase 0, you will divide and conquer by breaking this problem into smaller mini-phases that each build off of the previous one.

Mini-Phase 0A: Start creating the Game class

You can look at the header file game.h to see what public function the Game class is.

You should create the `run()` function. This contains the loop that runs the game. For now, your loop that runs the game should just be:

```
while (true) {
      return;
}
```

Mini-Phase 0B: Print the board and place the boulders

When an instance of your Game class is created, it automatically creates a 2-D array called `board` and an array of Boulders called `boulders`. You can see these private members in the Game header file `game.h`. In the `run()` function, you should clear the board and print each boulder. Remember, you don't print all the boulders, just 3 (for now). You **must** do this by calling `place_on_board()` (which in the Boulder class). But, you can create a helper function (or multiple) and then call it from `run()`. These can private member functions of the Game class.

Mini-Phase 0C: Print the board and place the boulders and the dots

You should create the Dot class and implement the functions described in the Dot header file `dot.h`. The Dot constructor should initialize the private members of the Dot class. The row and column can just be initialized to 0, because the actual location will be set by the `set_random_location()` function.

Uncomment the line `Dot dots[NUM_DOTS];` from game.h.

Now, when an instance of your Game class is created, it automatically creates an array of Dots called `dots`. You can see this private member in the Game header file `game.h`. In the `run()` function, you should initialize each Dot by setting the its eaten attribute `eaten` to `false` using the `set_eaten()` function (which is in the Dot class), and setting the locations of the Dots using the `set_random_location()` function (which is also in the Dot class). Then, you should print each dot on the board. Remember, you don't print all the boulders, just 15 (for now). You **must** do this by calling `place_on_board()` (which in the Dot class) for each dot in the `dots` array. You can create a helper function (or multiple) and then call it from `run()`. These can private member functions of the Game class.

Phase 1: Due April 3, 2016, 11:59 PM

Summary:

- Your program must play one level with 3 boulders, 15 dots, 4 ghosts, and Pacman
- Pacman moves left, right, up and down.
- Pacman can wrap around the board.
- When Pacman eats a dot, it vanishes from the board.
- When Pacman eats all the dots, the game is over and the player wins.
- Ghosts must appear on the board, but they can be stationary (frozen).

For Phase 1, you will divide and conquer by breaking this problem into smaller mini-phases that each build off of the previous one. This will incorporate your progress was Phase 0.

Mini-Phase 1A: Introduce Pacman

After completing the previous phases, your program should print the board with the blocks and dots placed randomly on the board. Now, we are going to introduce Pacman.

You should create the Pacman class and implement the functions described in the Pacman header file `pacman.h`. The Pacman constructor should initialize the private members of the Pacman class (it can call `center()`, which is also in the Pacman class, to initialize the row and column). Remember that Pacman starts in the center of the board (You can and should use the `center()` function in the Pacman class to do this). The other functions are explained in the header file. Remember that `move()` sets `alive` to false if Pacman moved into a Ghost, and returns true if Pacman ate a Dot.

You should include Pacman as part of the game. Uncomment the line `Pacman p;` from game.h.

Now, when an instance of your Game class is created, it automatically creates an Pacman called `p`. You can see this private member in the Game header file `game.h`. You should change your loop inside `run()` so that it follows this pattern:

1. Clear board
2. Place boulders on board
3. Place the non-eaten dots on board
4. Place Pacman on board
5. Get an input from the user
6. Move Pacman by running `move()` (which is in the Pacman class)
7. Repeat

If Pacman eats a Dot, Pacman's `move()` function returns true. If this happens, you should set the associated Dot in the `dots` array to have been eaten.

To help you out, you can create a helper function (or multiple) and then call it from `run()`. These can private member functions of the Game class.

Mini-Phase 1B: Winning

Instead of your loop in run() running forever, change the condition so that the loop stops if all the Dots are eaten, and then the program prints a winning message.

Mini-Phase 1C: Stationary Ghosts

You should create the Ghost class and implement the functions described in the Ghost header file `ghost.h`. The Ghost constructor should initialize the private members of the Ghost class. The other functions are explained in the header file.

Uncomment the line `Ghost ghosts[NUM_GHOSTS];` from game.h.

Now, when an instance of your Game class is created, it automatically creates an Ghost array called `ghosts`. You can see this private member in the Game header file `game.h`. In the `run()` function, you should initialize all the ghosts to corners using the `set_location()` function (which is inside the Ghost class). You should change your loop inside `run()` so that after Pacman is placed on the board, the ghosts are placed on the board as well (using the `place_on_board()` function in the Ghost class). You can create a helper function (or multiple) and then call it from `run()`. These can private member functions of the Game class.

Now that you have all your classes written, you can compile using the given Makefile. In your terminal, just run `make`.

Mini-Phase 1D: Losing

Change the condition so that the loop stops Pacman if Pacman is no longer alive (using the `is_alive()` function in in the Pacman class), and then the program prints a losing message.

Phase 2: Due April 6, 2016, 11:59 PM

Summary:

- Your program must play through ten levels to win the game. On each level, the right number of dots/boulders should be printed on the board (see the rules for details).
- Your program must keep score (see the rules for details).
- Ghosts can still remain stationary.

Just like Phases 0 and 1, you should divide and conquer by breaking this problem into smaller mini-phases that each build off of the previous one.

Phase 3: Due April 8, 2016, 11:59 PM

Summary:

- Ghosts must move toward Pacman. Because it is too hard otherwise, the ghosts move once for every `GHOST_FREQ` moves the user makes (hint: use Pacman's `get_num_moves()` function). `GHOST_FREQ` is defined in `constants.h`.

Suggested Ghost Algorithm:

2. Find the difference in the columns and in the rows of the Pac-Man and the ghost (think about using the abs() function here)
3. If the difference of the columns is greater than the difference of the rows, increment or decrement the column location of the ghost to move closer to the Pac-Man
4. If the difference of the rows is greater than the difference of the columns, increment or decrement the row location of the ghost to move closer to the Pac-Man
5. If the location the ghost wants to move is occupied by another ghost, then don't move.

Above and Beyond

There are a lot of opportunities to go above and beyond in this assignment. Here are some that we came up with, but feel free to be imaginative and come up with your own!

- In the real Pacman game, power-pellets temporarily give Pacman the ability to eat ghosts. Implement power-pellets.
- The Ghost AI is rudimentary. Change it to be better.
- The real Pacman game has moving fruit that pac-man can eat -- implement this.
- When companies feel that their product is being duplicated without license, they often resort to legal action. Create a Pacman game that is so realistic that you receive a cease and desist letter from Namco.
- Games are more fun with friends. Implement 2-Player Pacman.

Providing

Phase 0:

```
provide comp11 proj1a *.h boulder.cpp dot.cpp game.cpp
```

Phase 1:

```
provide comp11 proj1b *.h boulder.cpp dot.cpp game.cpp ghost.cpp pacman.cpp
```

Phase 2:

```
provide comp11 proj1c *.h *.cpp
```

Phase 3:

```
provide comp11 proj1d *.h *.cpp
```