Project: CP's PACMAN[*†]

Sibt ul Hussain

May 4, 2014

Design Deadline (Class Diagrams + Drawing): May 16, 2014 before 23h30
Deadline: May 13, 2014 before 23h30

Attention

- Make sure that you read and understand each and every instruction. If you have any questions or comments you are encouraged to discuss your problems with your colleagues (and instructors) on Piazza.
- Plagiarism is strongly forbidden and will be very strongly punished. If we find that you have copied from someone else or someone else has copied from you (with or without your knowledge) both of you will be punished. You will be awarded (straight zero in the project — which can eventually result in your failure) and appropriate action as recommended by the Disciplinary Committee (DC can even award a straight F in the subject) will be taken.
- **Try to understand and do the project yourself even if you are not able to complete the project. Note that you will be mainly awarded on your effort not on the basis whether you have completed the project or not.**
- Divide and conquer: since you have around 10 days so you are recommended to divide the complete task in manageable subtasks. We recommend to complete the drawing and design (*i.e.* number of classes and their relationships) phase as quickly as possible and then focus on the intelligence phase.

[*] Play Online:`https://www.google.com/doodles/30th-anniversary-of-pac-man`
[†] Complete set of instruction on how to build it can be found here: `http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior`

- **Before writing even one line of code, you must design your final project. This process will require you to break down and outline your program into classes, design your data structure(s), clarify the major functionality of your program, and pseudocode important methods. After designing your program, you will find that writing the program is a much simpler process.**
- **No Marks will be given if you do not submit your class diagram and if you do not use the object oriented design principles you have learned during the course.**
- *Imagination Powers: Use your imaginative powers to make this as interesting and appealing as you can think of. An excellent solution can get you bonus marks*

Goals: In this project you will build a 2D game (CP's PACMAN – see Figure 1) using the techniques learned during the course. The major goal of this project is to consolidate the things you have learned during the course. In this respect it is requested to

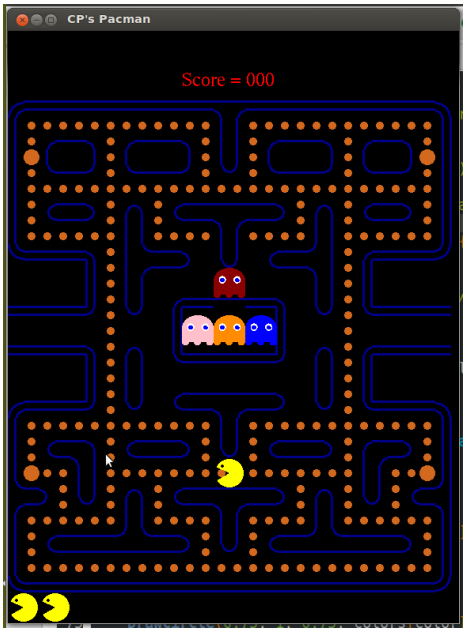

Figure 1: A screen shot of the game.

1 Instructions

We provide complete skeleton with all the basic drawing functions (can be located in util.h and util.cpp) needed in project with detailed instructions and documentation. In other words all you need to know for building the game is provided. Your main task will be to understand the main design of game

and implement it. Complete set of instructions on how to build the pacman are given on following url: `http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior` (the webpage is included for offline viewing) However, before proceeding with code writing you will need to install some required libraries.

1.1 Installing libraries on Linux (Ubuntu)

You can install libraries either from the Ubuntu software center or from command line. We recommend command line and provide the file "install-libraries.sh" to automate the complete installation procedure. To install libraries:

1. Simply run the terminal and go to directory which contains the file downloaded file "install-libraries.sh".
2. Run the command
```
bash install-libraries.sh
```
3. Provide the password and wait for the libraries to be installed. If you get an error that libglew1.6-dev cannot be found, try installing an older version, such as libglew1.5-dev by issuing following on command line
```
sudo apt-get install libglew1.5-dev
```
4. If you have any other flavour of Linux. You can follow similar procedure to install "OpenGL" libraries.

1.2 Compiling and Executing

To compile the game (skeleton) each time you will be using "g++". However to automate the compilation and linking process we use a program "make". Make takes as an input a file containing the names of files to compile and libraries to link. This file is named as "Makefile" in the game folder and contains the detail of all the libraries that game uses and need to linked.

So each time you need to compile and link your program (game) you will be simply calling the "make" utility in the game directory on the terminal to perform the compilation and linking.

```
make
```

That's it if there are no errors you will have your game executable (on running you will see three shapes on your screen). Otherwise try to remove the pointed syntax errors and repeat the make procedure.

1.3 Drawing Board and Shapes

Canvas Since we will be building 2D games, our first step towards building any game will be to define a canvas (our 2D world or 2D coordinate space in number of horizontal and vertical pixels) for drawing the game objects (in our case ball, board and bricks). For defining the canvas size you will be using (calling) the function "SetCanvas" (see below) and providing two parameters to set the drawing-world width and height in pixels.

```
/* Function sets canvas size (drawing area) in pixels...
 *  that is what dimensions (x and y) your game will have
 *  Note that the bottom-left coordinate has value (0,0)
 *  and top-right coordinate has value (width-1,height-1).
 *  To draw any object you will need to specify its location
 * */
void SetCanvasSize(int width, int height)
```

Drawing Primitives Once we have defined the canvas our next goal will be to draw the game board and its characters using basic drawing primitives. For drawing each object we will need to specify its constituent point's locations (x & y coordinates) in 2D canvas space and its size. You will only need lines, circles, curves (toruses), rounded rectangles and triangles as drawing primitives to draw the complete board, pacman and ghosts.

For this purpose, skeleton code already include functions for drawing lines, circles, curves, rounded rectangles (see below), traingles at specified location.

Recall that a line needs two vertices (points) where as a triangle needs three vertices so to draw these primitives we will need to provide these vertices (points) locations along with shape's color – *c.f.* Figure 2. Skeleton already provides a list of ≈ 140 colors which can be used for coloring different shapes – note that each color is combinations of three individual components red, green and blue.

Drawing Board Initially it might seem drawing and managing the board is extremely difficult however this difficulty can be overcome using a very simple trick of divide and conquer. The trick revolve around the idea of the board being split into tiles. "Tile" or "cell" in this context refers to an 8×8 – you can use any tile size as you wish – pixel square on the screen. PacMan's screen resolution is 224×288, so this gives us a total board size of $28x36$ tiles. Since we will be working independent of pixel units, so we can define tile size in our coordinates units, for instance we can divide the board in 8×8[1] units – *c.f.* Figure 3. So drawing and managing the board will require these two steps:

1. Splitting the board in tiles.
2. Finding and storing what shape to draw in each tile.

Once we have divided the boards into tiles our job reduces to finding what lies in each tile *i.e.* what primitive shape we need to draw in each tile. We can record this information in an offline table and then can loop over this table to draw each primitive. We can further simplify our task by defining an `enum` environment to assign constant names (integers) to these primitives and then build table of these primitives. Figure 4 shows an example on how can we draw some part of the board using a 2D offline table. Complete code can be found in the skeleton.

Following similar lines you can draw the complete board. Note that your system must follow object oriented design principles.

Remember that you can do your drawing only in the `Display()` function, that is only those objects will be drawn on the canvas that are mentioned inside

[1] use variables to store the tile size

```
// Drawing functions provided in the skeleton code

/* To draw a triangle we need three vertices with each
 *  vertex having 2-coordinates [x, y] and a color for the
 *  triangle.
 *  This function takes 4 arguments first three arguments
 *  (3 vertices + 1 color) to draw the triangle with the
 *  given color.
 * */
void DrawTriangle(int x1, int y1, int x2, int y2, int x3,
          int y3,float color[]/*three
                component color vector*/)

// Function draws a circle of given radius and color at the
// given point location (sx,sy).
void DrawCircle(float x, float y, float radius,
                float*color);

// Function draws a circular curve of given radius
void Torus2d(int x /*Starting position x*/,
        int y /*Starting position Y*/,
        float angle, // starting angle in degrees
        float length, // length of arc in degrees, >0
        float radius, // inner radius, >0
        float width,  // width of torus, >0
        unsigned int samples=60,// number of circle samples, >=3
        float *color = NULL);

// Function draws a line between point P1(x1,y1) and P2(x2,y2)
// of given width and colour
void DrawLine(int x1, int y1, int x2, int y2,
            int lwidth = 3, float *color =NULL);

// Function draws a rectangle with rounded corners
at given x,y coordinates
void DrawRoundRect(float x, float y, float width,
                float height,
                float* color = 0,
                float radius = 0.0/*corner radius*/);

// Function draws a string at given x,y coordinates
void DrawString(int x, int y, const string& str,
        float * color = NULL);
```

Figure 2: A set of functions for drawing primitive shapes.

the Display function. This `Display` function is automatically called by the graphics library whenever the contents of the canvas (window) will need to be

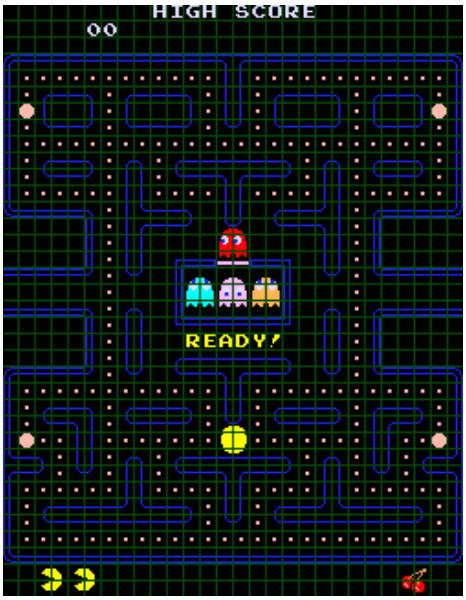

Figure 3: Pacman board divdided into grids. Note that once we have divided the board into a grid we can build a 2D table in which we can record the contents of each tile, *i.e.* what shape it contains and at what location. Given the contents table we can loop over this table and draw the complete board

```
// A simple example of board
enum BoardParts {
        NILL, // Prohibitive Empty space
        TLC, // Left corner top
        TRC, //Right corner top
        BLC, // Left corner bottom
        BRC, //Right corner bottom
        HL, // Horizontal line
        VL, // Vertical line
        PEBB, // Pebbles
};
const int BOARD_X = 10;
const int BOARD_Y = 5;
static int board_array[BOARD_Y][BOARD_X] = {
{ PEBB, PEBB, PEBB, BRC, BLC, PEBB, VL, VL, PEBB, PEBB},
{ PEBB, PEBB, PEBB, VL, VL, PEBB, PEBB, PEBB, PEBB, PEBB },
{ PEBB, PEBB, PEBB, VL, VL, PEBB, PEBB, PEBB, PEBB, PEBB },
{ BRC, HL, HL, TLC, TRC, HL, HL, HL, HL, BLC },
{ TRC, HL, HL, HL, HL, HL, HL, HL, HL, TLC } };
```

Figure 4: Example code for generating some section of the board.

drawn *i.e.* when the window is initially opened, and likely when the window is raised above other windows and previously obscured areas are exposed, or when `glutPostRedisplay()` is explicitly called.

In short, `Display` function is called automatically by the library and all the things inside it are drawn. However whenever you need to redraw the canvas you can explicitly call the Display() function by calling the function `glutPostRedisplay()`. *For instance, you will calling the* `Display` *function whenever you wanted to animate (move) your objects; where first you will set the new positions of your objects and then call the* `glutPostRedisplay()` *to redraw the objects at their new positions. Also see the documentation of* `Timer` *function.*

1.4 Interaction With Game

For the interaction with your game you will be using arrow keys on your keyboard (you can use mouse and other keys as well). Each key on your keyboard have associated ASCII code. You will be making using of these ASCII codes to check which key is pressed and will take appropriate action corresponding the pushed key. *E.g.* to move the board right you will check for the pressed key if the pressed key is left arrow you will move the board left (change its position). Keyboard keys are divided in two main groups: printable characters (such as a, b, tab, *etc.*) and non-printable ones (such as arrow keys, ctrl, *etc.*). Graphics library will call your corresponding registered functions whenever any printable and non-printable key from your keyboard is pressed. In the skeleton code we have registered two different functions (see below) to graphics library. These two functions are called whenever either a printable or non-printable ASCII key is pressed (see the skeleton for complete documentation). Your main tasks here will be add all the necessary functionality needed to make the game work.

```
/*This function is called (automatically by library)
 * whenever any non-printable key (such as up-arrow,
 * down-arraw) is pressed from the keyboard
 *
 * You will have to add the necessary code here
 * when the arrow keys are pressed or any other key
 * is pressed...
 * This function has three argument variable key contains
 * the ASCII of the key pressed, while x and y tells the
 * program coordinates of mouse pointer when key was
 * pressed.
 * */
void NonPrintableKeys(int key, int x, int y)

/* This function is called (automatically by library)
 * whenever any printable key (such as x,b, enter, etc.)
 * is pressed from the keyboard
 * This function has three argument variable key contains
 * the ASCII of the key pressed, while x and y tells the
 * program coordinates of mouse pointer when key was
 * pressed.
 * */
void PrintableKeys(unsigned char key, int x, int y)
```

1.5 Collision Detection

2 Ghost Behaviour

For complete details on how the ghost behave and how to make them intelligent read the following webpage: `http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior`