This project creates an Mbed version of the classic Pac-Man video arcade game that was first released in Japan in 1980 by Namco and became wildly popular in the 80's. Google commemorated Pac-Man's 30th anniversary in 2010 with its first-ever fully interactive doodle, a playable version of the game with the Google logo in the maze.

In our version of the game, the player moves a yellow pacman through a maze lined with cookies (also known as "pellets" or "pac-dots"). As the pacman eats the cookies in its path, it collects points. Also moving about the maze are ghosts. If the pacman and a ghost run into each other, under normal conditions, the pacman loses a life. If the pacman loses all of its lives, it's Game Over! However, in addition to cookies, the maze contains super-cookies (also called "power pellets") which are larger than the rest. For a short period of time after eating one of these, the pacman becomes invincible and can kill any ghosts it runs into.

In this project, you will be given shell code that implements parts of the game and you will complete the rest. The portion given to you is the main game loop, initial maze datastructure and display, the pacman and ghost display features, and the ghost movement. You must complete the basic game functionality (described below) for 50 points. You can earn additional points by successfully implementing extra features to enhance the basic game (as described below). These are worth 5 points per extra feature, for a maximum of 50 additional points (10 additional features). So, to get a 100% on this project (25% for P2-1 and 75% for P2-2), your mbed program must support the basic game and 5 additional features. Supporting up to 5 more features will earn you up to 25 points extra credit. (So, the maximum total score of P2-1 and P2-2 combined is 125 points obtained by implementing 10 extra features.)

P2-2 (50 points): Basic Game Functionality

These features must be implemented and working correctly to earn the baseline 50 points:

Pacman movement: This uses the accelerometer to sense when the player tilts the circuit board in a given direction. Your program should use the data from the accelerometer to control the direction in which the pacman should move. Your program should keep track of where the pacman is in the maze and in what direction it is currently heading. It should use the map API functions to determine if the path is open/blocked ahead. It should update the display at each time step to animate the pacman's movement if there is not a maze wall blocking the movement. Note that the maze wraps around on the edges, so if the pacman goes off the screen through an

open path on one edge, it should reappear on the opposite edge, still heading in the same direction. (The uLCD is 128x128 pixels.)

Eating cookies: The maze is a 17x16 map of square locations (each location is called a "grid"). The status of any given grid in the maze is one of the following: Empty, Cookie, Wall, Super_Cookie. Whenever the pacman moves into a new location, your program should use the map API functions to lookup whether that location in the maze contains a cookie. If so, it should change the status of that location to Empty and keep track of how many cookies have been eaten. It should update the score (1 point/cookie).

Eating super-cookies: Whenever the pacman moves into a location containing a super-cookie, your program should change the status of that location to Empty, update the number of cookies eaten and the score (5 points/super- cookie). It should make the pacman invincible for a certain period of time (e.g., 5 seconds in level 1, 4 seconds in level 2, etc.). While the pacman is invincible, it can run into ghosts and not lose a life. Instead, the ghost dies and the score increases by 10 points.

Detecting run -in with a ghost and Game Over: Your program needs to determine whether the pacman and a ghost run into each other. If the pacman is not invincible (has not eaten a super-cookie recently), then the pacman loses a life. If there are no more lives remaining, the game ends with a "Game Over" message displayed on the uLCD screen. (The pacman should have 3 lives when the game starts.) Your program should use the ghost API functions.

Displaying number of pacman lives remaining: display the number of lives remaining on the uLCD screen's upper right corner.

Displaying score: display the number of points collected so far on the uLCD screen in the upper left corner.

Advancing levels: When the maze is clear of cookies and super-cookies, the game should advance to the next level. This should switch to a new maze, add a life, and shorten the invincibility period.

Force level advance: When two pushbuttons are pressed simultaneously, the game should advance to the next level. (This will make it easier to test and grade the "Advancing levels" feature.)

P2-2 (up to 50 points): Extra Features

You can implement a subset of these features to earn an additional 5 points per feature, up to a maximum of 50 points:

More dramatic death of pacman/ghost – use more complex visual effects to animate the ghost/pacman's loss of life.

Show number of lives remaining in icons – rather than using a number.

Add ephemeral fruit – Put fruit in the maze that only lasts for a few seconds. If the pacman eats the fruit, it collects bonus points.

Add a more complex "super" ghost whose implementation makes use of the linked list. A super ghost is a series of normal ghosts concatenated to make a body that is longer than a normal ghost and snakes around the maze. The head walks randomly while the rest of the body follows behind. The linked list keeps track of the positions of the ghosts within the super ghost.

Add new hardware to do something interesting, such as using the RGB LED to indicate some status information with color (http://developer.mbed.org/users/4180_1/notebook/rgb-leds/).

Include a Game Menu for configuring the game (e.g. starting it at some level: Easy/Medium/Hard).

Keep track of game history and show in interesting way (e.g., highest score so far or fastest time to clear a maze) – this requires that you reset the game using a pushbutton without using the mbed's reset button which reinitializes the entire program. When game ends, generate an interesting animation/sound (e.g., a triumphant one when you get a new high score).

Make the ghost(s) smarter – make the ghosts use more directed navigation strategies to chase or ambush the pacman. (The original game had 4 ghosts, "Blinky", "Pinky", "Inky", and "Clyde" that had different personalities.)

Add sound effects – be creative. These could accompany eating a super- cookie, advancing to a new level, gaining a new life, hitting a new high score, etc. Be careful not to call playSound (which accesses wav files from the sd card) on every iteration of the main game loop because this will slow it down and interfere with the interactive game playing.

As an alternative for short sounds or playing single notes, there is documentation on "Playing a Song with PWM using Timer Interrupts" here: http://developer.mbed.org/users/4180_1/notebook/using-a-speaker-for-audio-output/
Hello world song player code URL is http://developer.mbed.org/users/4180_1/code/song_demo_PWM/
Video of a media player: http://developer.mbed.org/users/4180_1/notebook/mpod---an-mbed -media-player/

Use pushbuttons to create a new feature that can be used a limited number of times: for example, a hypertransport that allows the pacman to hop from one part of the maze to another.

To get more ideas, Google "Pac-Man" to find playable versions of the original classic game (including the Google Pac-Man Doodle). If the feature you would like to implement is not mentioned here, that's great, but be sure to check with an instructor/TA to see if it would count.

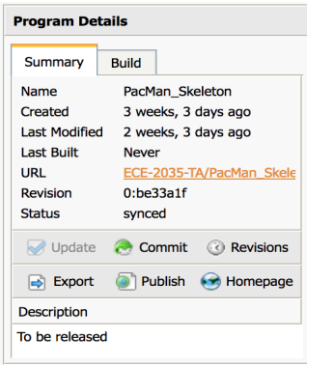
Shell Code and API Support

The project shell code is available here. Click the "Import this program" button (upper right) to bring this project into your own account. The API files can be found in the "Files" folder under the imported program. Add your own doubly linked list c code finished in the previous project into the program in order to compile successfully.
Also, a feature checklist file named `P2.checklist.txt` is available on T-square. Put an X before each feature in the checklist that your program implements and that you will demonstrate to the TA.

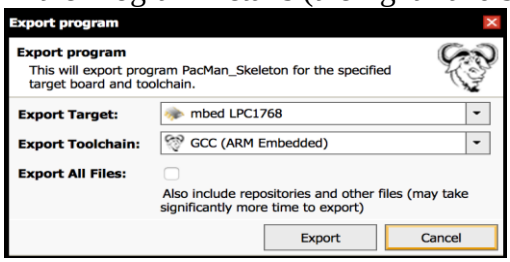
Project Submission

In order for your solution to be properly received and graded, there are a few requirements.

1. Create a .zip archive of your project using the following steps:



- In the Program Details (the right-hand side in the mbed compiler), click Export



- A box will pop up. Choose "GCC (ARM Embedded)" in the Export Toolchain menu. Try to reimport the exported file to mbed compiler and see if it will work.
- Select “Export” and another dialog box will pop up that will allow you to save the file. Name this archive **`P2.zip`**.
- Upload **`P2.zip`** and **`P2-checklist.txt`** to T-square before the scheduled due date, **5:00pm on Mon, 17 April 2017** (with a one hour grace period).

2. After your project zip file is uploaded to T-square, meet with a TA and demo your game on **your** hardware. While the TA is watching, you will download the files you submitted from T-square, compile them in the mbed cloud compiler, download the executable to your mbed, and then demonstrate all of the features of your game. Bring a printout of the checklist you submitted showing which features you successfully implemented so that the TA can confirm them. This must be completed by **Friday, 21 April 2017.**

You should design, implement, and test your own code. There are many, many ways to code this project, and many different possibilities for timing, difficulty, responsiveness and general feel of the game. Your project should represent your interpretation of how the game should feel and play. Any submitted project containing code (other than the provided framework code and mbed libraries) not fully created and debugged by the student constitutes academic misconduct.

Project 2 Grading: The project weighting will be determined as follows:

part	*description*	*due date*	*percent*
	Pac-Man Program		
P2-1	Doubly Linked List	Monday, 3 Apr 2017	25
P2-2	Baseline features	Monday, 17 Apr 2017	50
P2-2	Advanced Features		50
P2-2	Demo to TA for check-off	Mon-Fri, 17-21 Apr 2017	
	total		125/100 maximum

Extra Extra-Credit: Here's another chance to earn extra credit (and the admiration of your fellow students). Create a short (< 1 minute) video clip highlighting the coolest feature(s) of your mbed project. If you enter a high quality video, you will earn an extra credit point on your overall course grade. (This is particularly helpful if you are on a letter grade boundary).

Your video clip must clearly identify the features that you are highlighting. To submit your entry, save your video clip as a **`.mov`** or **`.mp4`** file named "**`P2clip.mov`**" or "**`P2clip.mp4`**" and upload it by **Friday, 21 April 2017** to T-square under Assignments > "Missile Command Highlights". We'll show a highlight reel of the video clips in lecture on the last day of classes.

References

1. Map API (for accessing the maze): http://developer.mbed.org/teams/ECE2035-Spring-2015-TA/code/Pacman/docs/fd547f008985/map__public_8h.html
2. Ghost API: http://developer.mbed.org/teams/ECE2035-Spring-2015-TA/code/Pacman/docs/fd547f008985/ghost_8h.html
3. Shell code: http://developer.mbed.org/teams/ECE2035-Spring-2015-TA/code/Pacman/
4. Mbed Pinout: http://mbed.org/nxp/lpc2368/quick-reference/
5. Mbed-NXP Pinout: http://mbed.org/users/synvox/notebook/lpc1768-pinout-with-labelled-mbed-pins/
6. NXP-LPC1768 User's Manual: http://www.nxp.com/documents/user_manual/UM