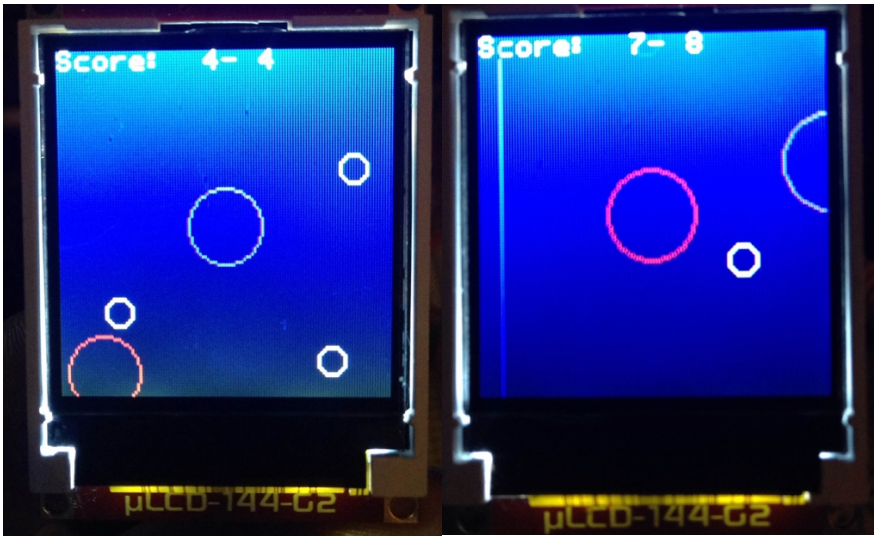
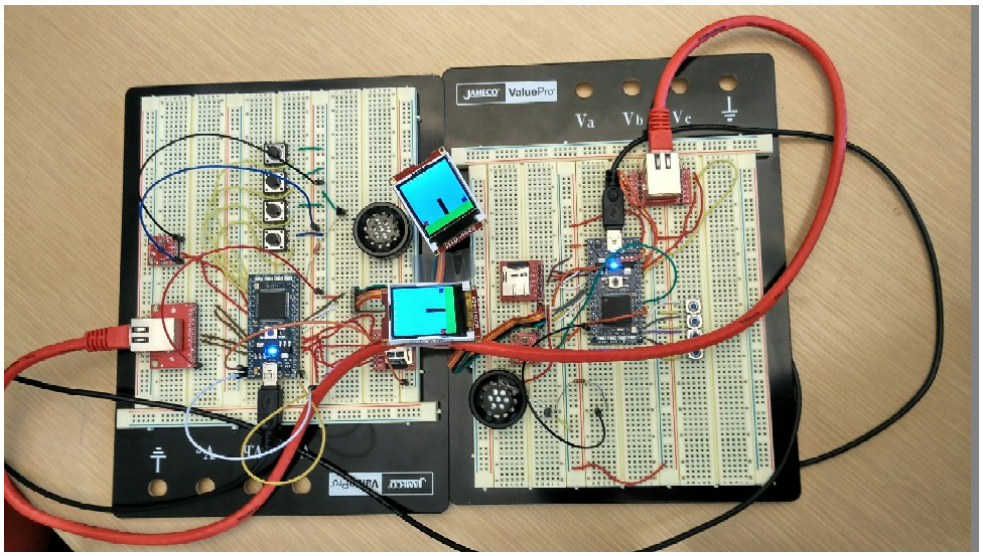
This project creates an Mbed version of an interactive game called **Agar.gt** similar to the popular game **Agar.io**. We recommend you try it out to get a good feel for this project.
In our version of the game, a player is associated with a blob, that interacts with other (non player (AI controlled) or second player) blobs and assorted objects in the game field of play. A blob can move in any direction with the goal of eating smaller blobs, and its (maximum) speed decreases as its mass increases. A blob gains mass by colliding with and eating smaller blobs in the field of play. The field of play is a large rectangular petri dish with at least 512 x 512 pixels. In this game, the player is not afforded a full view of the field of play. Instead, the 128 x 128 uLCD screen serves as a floating "window" into a larger world. In the base version of the game, if the player's blob is eaten by another blob, the game is over. The baseline game may be played in single player mode in which the other player is controlled by the game, and in two player mode, where the second player's mbed setup is remotely controlled over an Ethernet link.

To use the Ethernet link you have been provided with an Ethernet API called the Game Synchronizer. It allows you to connect your mbed to a second "slave" mbed over Ethernet and to draw basic graphics primitives to either or both screens.

There are a multitude of extra credit options available to you on this project (to be detailed later in the assignment description), and we recommend you start early so that you are able to take full advantage of them.



Example Agar.gt screens



Two mbed's connected via Ethernet in two-player mode

Game Rules

- Each player is able to see a view of the game world (displayed on its LCD screen) which contains their blob. This could be just always centered on their blob, or it could be something else.
- A player controls the direction of their blob will move by tilting their board. The game determines the direction by reading the accelerometer.
- A blob moves at a velocity that is inversely proportional to its mass (area). You are free to set the baseline velocity to enhance the feel of the game.
- If two blobs collide, the blob with greater mass will consume the blob with lesser mass, increasing its own mass correspondingly. We leave it to you to determine how dissimilar blobs must be for the smaller to be eaten by the larger.
- A blobs mass is represented by its area on the screen. That is all blobs have the same density.
- A player wins if its blob eats up the opponent's blob.
- On startup the world is randomly seeded with an odd number small food blobs. Food blobs float around the game world without interacting with each other. In the baseline game, player blobs grow by consuming food blobs.
- A blob cannot move off the edge of the screen. If a food blob reaches the edge of the game world, it bounces completely elastically (reverses direction without losing speed). Player blobs cannot move outside the bounding rectangle of the game world.

Implementation

In this project, you will be given shell code that implements parts of the game, and you will complete the rest. You must complete the basic game functionality (described below) for 75 points. You can earn additional points by successfully implementing and documenting 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, your mbed program must support the basic game and 5 additional features.

Basic Game Functionality

These features must be implemented, documented and working correctly to earn the baseline 75 points:

Game Entry Menu: At startup, a menu screen should appear on player 1's uLCD asking the player whether they wish to play in single- or multi-player mode. This will require access to player one's uLCD and push buttons on the mbed board. (The menu in the shell code always defaults to SINGLE_PLAYER mode.) Additional modes may be available depending on your particular implementation. For full credit, you must display reasonable effort to make the menu appealing to potential players. Be creative!

Supporting 1-Player/2-Player Game Control: Implement the game logic for the two modes of play (one or two player). Allow the players to move their blob around the game world, displaying their respective views on their respective screens. The game world must be at least four times the screen size in each dimension (The screen is 128x128, so the game world should be at least 512x512). When the blobs are sufficiently close, they will both appear on both screens and one will consume the other if they collide (unless they are the same mass, in which case they do not overlap). In single player mode, the opponent blob is under AI control. How intelligent you make the AI is up to you. In two-player mode, the Student side mbed program is the main program in charge. It computes what to do with the inputs and how to update both screens. The TA side mbed is "slaved" to the Student mbed. It does nothing but gather accelerometer and pushbutton inputs and send them to the Student side for processing. The student side sends back screen display commands that the TA side follows to update its screen. The TA side code is provided in an executable named **`TA.bin`**. To play your game in 2-player mode with another student, load one mbed with the executable for your game and load the other student's mbed with the TA side executable. Connect the two boards together using your Ethernet cable and reset both mbeds to play!

(A demo version student side program **`Student.bin`** is provided on T-square, along with **`TA.bin`** so you can get an initial idea of the game play.)

Blob movement: In the basic game, a player blob moves in any direction to eat food and try to grow as big as possible. Your program should use the data from the accelerometer to control the blob. (Tilting the board left/right moves the blob left/right; tilting the board forward/backward moves the blob up/down.) In order for the blob to fully interact with its environment, the blob must include collision detection with objects in the scene as well as the edges of the world. No blob should be permitted to leave the bounding rectangle of the game world.

Blob Interaction with objects around: The blob interacts with the objects around it in a specific manner. In the version of the game provided to you, the food balls around the blob are circles of a smaller size which can be eaten up by the blob. For each food ball eaten by either blob, it grows bigger by a fixed amount.

Landscape: Enhance the background landscape by adding features to it. Features may include additional obstacle blocks, changing initial blob positions, and other background scenery. The blobs must always be visible over the background.

Score: Display a small indicator somewhere on the screen indicating how much food each player has consumed.

Sound effects: Your game should play a fun sound at the beginning when the menu screen appears and a game over sound at the end of the game. Make an interesting sound when a blob eats another blob. The sound should distinguish between player-food interactions and player-player interactions. Be creative!

Game Over: Your program needs to determine and display who won the game. Include some visual effect and text notifying the player that the game is over. As with the game menu, we encourage you to be creative! Text on a plain background probably won't cut it.

Extra Features

ONCE ALL OF THE BASELINE FEATURES HAVE BEEN IMPLEMENTED, you can implement a subset of these features to earn an additional 5 points per feature, up to a maximum of 50 points:

Friendly Blob: Make player blobs "friendly". This means that a player will not eat other blobs with the same color as the player.

Improve Gameplay: Come up with a creative new game mode. For example, a game mode where your goal is to avoid touching any other blob for as long as possible.

Use pushbuttons to create a new feature or powerup that can be used for a *limited total time*. Think outside the box! Examples include the following:

> **Acceleration Power Up:** Create a special power to that allows a blob to move faster for a fixed amount of time after the press of a button.
> **Invisibility Power Up:** Create a temporary invisibility feature that would allow a player to disappear on the opponent's screen.
> **Ink Power Up:** Create a temporary black screen for opponent. This feature could be won by a player who is losing to help them catch up to the winning player.
> **Split Power Up:** Create a powerup that allows a player to split their blob into two halves that move faster together.

Monster Blob: This is the cool one. Create a special blob that does something exciting. For example, a black hole blob that sucks in any player that gets too close! Or a blob that constantly trails your blob while trying to eat you. This is a good opportunity to program a cool AI to demonstrate your ability. (Keep in mind that our goal is to reward you for your effort.)

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/).
You can use different color of LED to indicate different situations, for example:

1. Turn the LED Red indicating danger if the opponent blob enters your view.

2. Turn the LED Blue indicating safe if the opponent blob is not in your view.

Difficulty Levels: for configuring the game (e.g. starting it at some level: Easy/Medium/Hard).

Create a Non-Rectangular Game World: Make the game world circular or some shape that is more interesting than a rectangle.

Keep track of game history and show in interesting way (e.g., Highscore, 2-player game best of 3 rounds) – this requires that you reset the game using a pushbutton without using the mbed's reset button which reinitializes the entire program or saving the state of the game history so that it can be reloaded when the mbed is reset.

Enhanced Graphics: Create a dramatic blob-consumption animation (or something else!).

Adding direction arrow: Add a small direction arrow near your blob to make the game more interactive. The direction arrow points in the direction your blob is moving. Alternatively, add a compass to the corner of the screen that always points to your opponent.

Add sound effects – be creative. These could accompany advancing to a new level, gaining or losing a life, winning a match, etc. Since the code already uses the MBED real-time OS, you can create a thread for playsound( ) to minimize its interference with gameplay. 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/

Other Feature Ideas: If you have an idea for a feature along the lines of the ones listed above, ask a GTA (Rudra or Jordan) or Course Instructor whether it will qualify. Our emails are available on the class site.

If you think it's really cool...
Seriously Impress Jordan Ford (5 to 10pts.): Subject to final approval by a professor, up to 10 points will be awarded for effort which truly exceeds expectations. Email jford38@gatech.edu to ask if your idea qualifies. Things that will impress me include

- Create a really sophisticated blob AI.
- Use the push-buttons to allow players to zoom in and out. This could get hard!
- Do something to allow the game to operate at a higher framerate with less flashing. Make the game smoother. Allow filled circles that don't flash.
- Improvements/new features for the Game Synchronizer (a BLIT instruction would be cool. Encrypting/Decrypting the ethernet packets would be cool. Etc.)
- Find (and fix!) a major bug. (For example, the Missing Internal Function message that keeps appearing on the uLCD screen.)
- Something even cooler! Seriously. Do something that interests you, and explain why it should interest me. Teach me something!

Shell Code

The shell code is available at:
https://developer.mbed.org/teams/ECE2035-Spring-2015-TA/code/ECE2035_Agar_Shell/
Click the "Import this program" button (upper right) to bring this project into your own account.

Where to begin?

We recommend starting by reading the shell code and comments, and the API document. Then start small, say create and move a blob. A key concept is the notion of a frame cycle. In each cycle or interval of time, the program must compute what will happen next for each object in the game based on the current game state and inputs, and then make those updates. You will need to "undraw" everything from the previous frame before you redraw the blobs in the next frame.

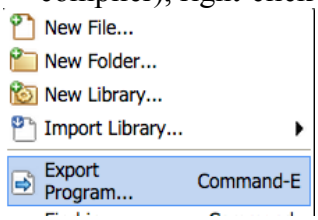
Checklist

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.

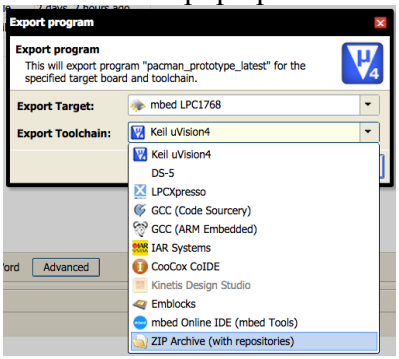
Project Submission

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

1. Compile your project and name the executable **`P2.bin`**.
2. Create a **`.zip`** archive of your project using the following steps:
   - In the Program Workspace (the left hand vertical scrolling window in the mbed compiler), right-click the project name of your project and select "Export Program..."



   - A box will pop up. Choose "ZIP Archive" in the Export Toolchain menu:



   - 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`**, **`P2.bin`**, and **`P2-checklist.txt`** to T-square before the scheduled due date, **5:00 pm on Monday, 18 April 2016** (with a one hour grace period).
3. 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 executable you submitted

from T-square 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, 22 April 2016.**

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 Grading - The project weighting will be determined as follows:

description	*percent*
Basic program functionality	
Technique, style, comments	15
Baseline features	60
Advanced Features	50
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 Agar.gt 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 **Thursday 21 April 2016** to T-square under Assignments > "Agar.gt Highlights".
We'll show a highlight reel of the video clips in lecture on the last day of classes with special recognition going to the top entries.

References

1. Shell code: https://developer.mbed.org/teams/ECE2035-Spring-2015-TA/code/ECE2035_Agar_Shell/
2. Mbed Pinout: http://mbed.org/nxp/lpc2368/quick-reference/
3. Mbed-NXP Pinout: http://mbed.org/users/synvox/notebook/lpc1768-pinout-with-labelled-mbed-pins/
4. NXP-LPC1768 User's Manual: http://www.nxp.com/documents/user_manual/UM10360.pdf