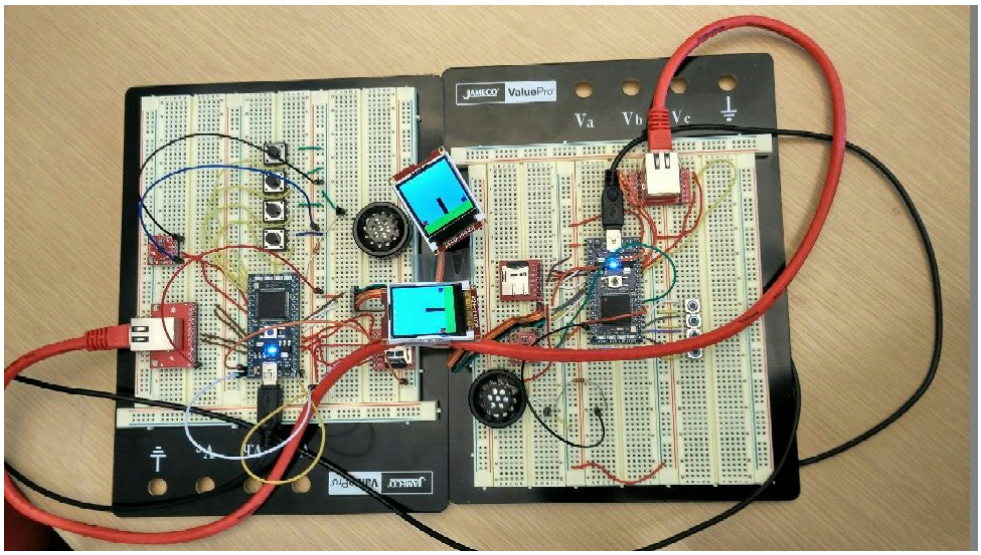
This project creates an Mbed version of a turn-based game called **Dueling Tanks**. In our version of the game, players control one of two tanks and aim to blow each other up. Only one player has control at a time, meaning that a player can move their tank, aim and fire only when it is their turn. When it is not their turn, the player is unable to move their tank. Once a tank has been hit by another tank, it blows up and the game is over. In the interest of fairness, both tanks must be the same size.

Dueling Tanks on microLCDs



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

Game Rules

- Each turn, a player gets one shot at their opponent.
- The control of the game is then transferred to the second player, who gets to shoot back.
- This repeats until one player has destroyed the other player's tank.
- Players can destroy the opposing tank (or their own) by executing a direct hit on a tank.

- A player wins if their opponent's tank is destroyed before their own tank.
- During their turn, a player can move their tank left and right prior to taking a shot at the opponent. The tank cannot drive past obstacles or off the edge of the screen.
- After taking their shot, the player is frozen and must await the return shot from the second player.

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 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 and working correctly to earn the baseline 75 points:

Game Entry Menu: At startup, a menu screen should appear 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.)

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 take turns: accept inputs from each player and compute and display their effects on the scene. In 1-player mode, the user will control both tanks, one at a time, so the accelerometer and pushbutton inputs for both tanks will come from the same mbed board but applied to alternating tanks. In 2-player mode, the accelerometer and pushbutton inputs will alternately come from the mbed board running the main program (Student side) and then from the Ethernet connected mbed board (TA side). The Student side mbed program is the main program in charge: computing what to do with the inputs and how to update the screen. The TA side mbed is simply gathering accelerometer and pushbutton inputs and sending 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.)

Tank movement: In the basic game, the tank moves left and right and aim its barrel up and down. Your program should use the data from the accelerometer to control the tank and its barrel during its turn. The accelerometer is also used to control the tank barrel angle. (Tilting the board left/right moves the tank left/right; tilting the board forward/backward increases/decreases the angle of the barrel.) In order for the tank to fully interact with its environment, the tank must

include collision detection with objects in the scene. The tank must be coded such that it cannot run through walls or off screen.

Bullet Launch and Trajectory: A button push fires the bullet. The firing of the tank cannon barrel requires the implementation of projectile motion to compute the bullet trajectory. (These equations are given in the Projectile Motion section below.) Additionally, explosion visuals must be implemented whenever the bullet collides with a solid object in the environment or a tank. (This is up to you, but in the baseline, it is implemented as a sky colored circle drawn at the point of impact.)

Landscape: Enhance the background landscape by adding features to it. Features may include additional obstacle blocks, changing initial tank positions, and different colors in the background. Color enhancement should avoid the sky blue or tank colors, as using these color values will cause errors if collisions are detected based on color of the pixel intercepted by the moving tank or bullet.

Turn Indicator: Display a small indicator somewhere on the screen indicating which player's turn it is (e.g., a triangle on the left or right side).

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. It should also make an explosion sound when the cannon ball hits a tank or solid object and blows it up.

Game Over: Your program needs to determine whether either tank has been hit, and therefore destroyed. (In default game rules, a tank cannot withstand a hit from a bullet, so if a tank is hit once, the game is over.) Make sure to include some visual effect and text notifying the player that the game is over.

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:

Bouncy Bullet: Make the fired bullet bounce off obstacles other than the tank until its velocity decays below a threshold value (student specified).

Show number of lives remaining in icons: Improved gameplay modes may include multiple lives, rounds and a scorecard. Display hearts in one of the corners of the uLCD to show the number of lives each player has left.

Use pushbuttons to create a new feature/powerup that can be used a *limited total number of frames*. Examples include the following:

> **Temporary Shield Power Up:** Create a shield that allows a tank immunity.
>
> **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.
>
> **Steal Power Up:** Create a powerup that allows a player to steal the other player's power up.

Hover Power Up: allows a tank to rise vertically and float above the landscape to get a better shot.

Special Bullet: Change bullet shape or function.

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/).

Enhanced Entry 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., fewest shots to win, 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: More dramatic shots or explosions.

Adding changing elevations to landscape: add hills and allow tank to climb or descend along the changing elevations. (Harder than you might think!) Ideas for how to do it: Describe your terrain as a piecewise differentiable function T(x). Find tank y-coord based on x-coord using T(x). Orient the tank along the terrain using T'(x). If you get that far, you'll get the credit. Don't worry too much about falling into craters left by tank bullets. (That's really hard!)

Robot opposing player in 1-player mode: make the other player autonomous so that it moves and shoots back at the player in 1-player mode (as opposed to the single player controlling both tanks one at a time). The AI must be nontrivial to earn credit. (That is, random moving and shooting doesn't count!).

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 (Keshav, 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

- Improvements/new features for the game_synchronizer (a BLIT instruction would be cool. Encrypting/Decrypting the ethernet packets would be cool. Etc.)
- Random Fractal Terrain http://www.gameprogrammer.com/fractal.html
- Something even cooler! Seriously. Do something that interests you, and explain why it should interest me. Teach me something!

Projectile Motion

Bullet trajectories should be calculated using the kinematic equations for projectiles. The kinematic equations for a projectile initially located at (x_0, y_0) are as follows:

$$y(t) = y_0 + v_y t - \tfrac{1}{2} g t^2$$

$$x(t) = x_0 + v_x t$$

where v_x and v_y are given by:

$$v_x = SPEED \cdot cos(ANGLE)$$

$$v_y = SPEED \cdot sin(ANGLE)$$

where SPEED and ANGLE are constant over the course of the bullet's trajectory. The <math.h> library contains implementations of sin( ) and cos( ).

IMPORTANT: To ensure your x and y values are compatible with the discrete nature of the game world, you should use the floor() function from <math.h> to truncate your x and y values.

$$y(t) = floor(y_0 + v_{0y} t - \tfrac{1}{2} g t^2)$$

$$x(t) = floor(x_0 + v_{0x} t)$$

Shell Code

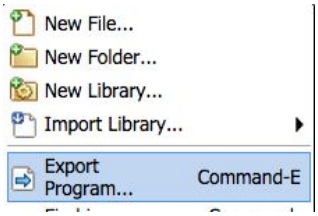
The shell code is available at:
https://developer.mbed.org/teams/ECE2035-Spring-2015-TA/code/2035_Tanks_Shell/
Click the "Import this program" button (upper right) to bring this project into your own account.

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.

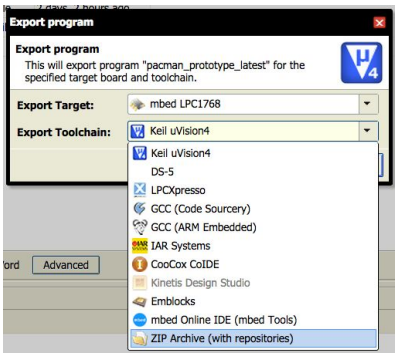
Project Submission

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 Friday, 13 November 2015** (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 **Monday, 23 November 2015.**

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 Dueling Tanks 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 **Sunday 29 Nov 2015** to T-square under Assignments > "Dueling Tanks 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/2035_Tanks_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