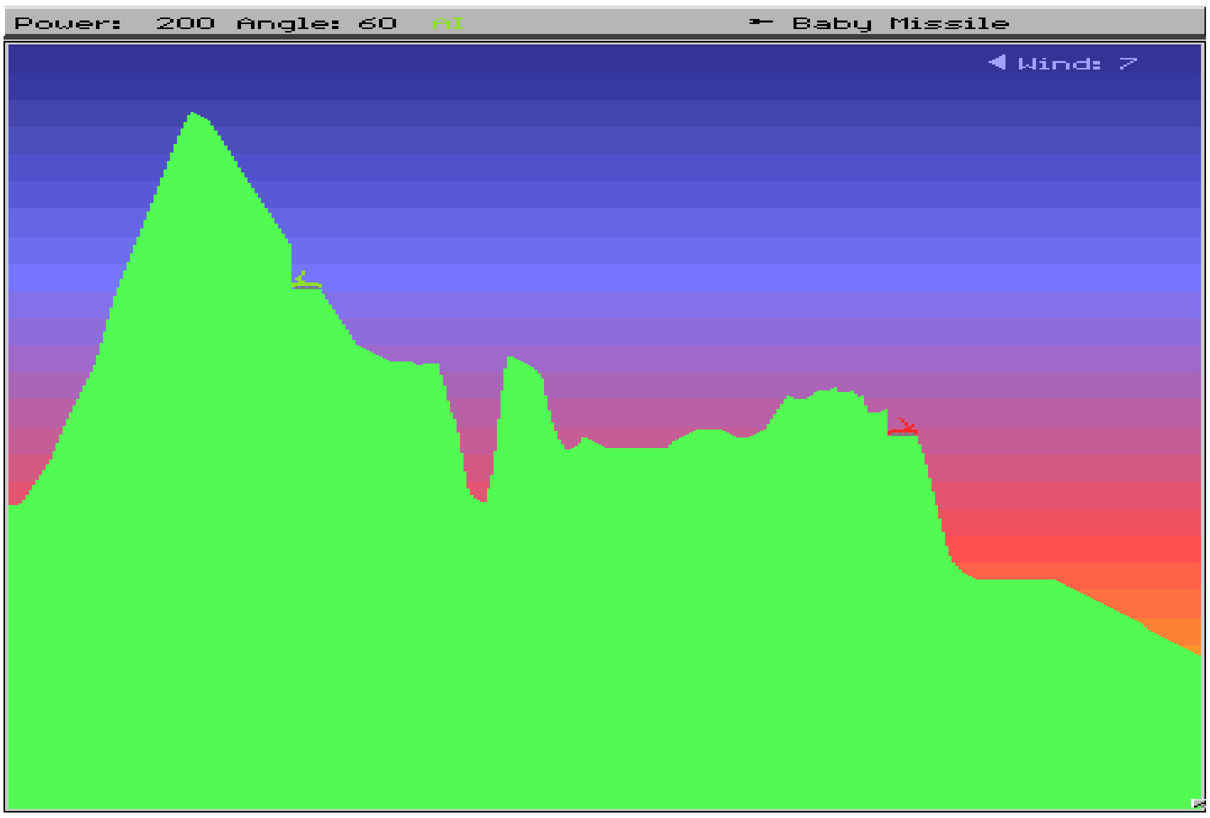
CAB202 Assignment 2

'Scorched Earth' on a Microcontroller: Part A

'Scorched Earth' is classic turn-based strategy game with the objective of destroying opposing tanks. The game was created in 1990 and first released in 1991 by Wendell Hicken. The final revision occurred in 2005 and the game is still available today (in the form of shareware - (http://www.whicken.com/scorch/index.html). Below is a screenshot from the game (if you wish to run it on your own computer use DOSBox - http://www.dosbox.com/download.php?main=1):



The basis of this assignment is a much simplified version of this game implemented on your Teensy hardware using the LCD display. The more challenging components of this assignment will extend the simplified version to be more challenging and closer to the actual game.

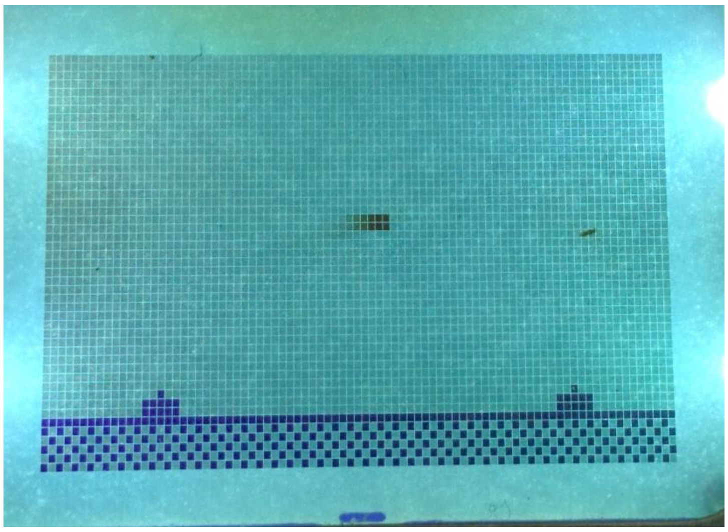
Task

For this assignment you must implement a version of the 'Scorched Earth' video game on the Teensy LCD board. The game is to be written in the C programming language. The implementation will make use of the LCD screen, buttons, LEDs, potentiometer, and ATMEGA32 chip. Your code must be written for the Teensy and **must compile** using the AVR-LIBC environment. You may not use any other libraries besides those which have been used in the prac classes.

The assignment will be done individually or in groups of two (no more than two under any circumstances). The exact specifications for your task (see the appropriate section) will depend on whether you are completing the assignment individually or in a group. You will demonstrate your game, explain the implementation, and be asked a variety of questions about your game during a designated marking timeslot in week 13. This session will examine your knowledge of your code and understanding of the operations you are demonstrating. If you cannot explain why and how parts of your code perform what they do, then you will **not receive marks** for that section.

Submission

The code used in your assignment will be submitted online to the AMS by **11:59pm Sunday the 19th of October** in preparation for week 13 marking sessions. Your code cannot be modified from this point onwards (a compiled **`*.hex`** of your submission will be downloaded and examined in the week 13 marking sessions – i.e. not what you bring in on your Teensy). All you will need to bring to this marking session is your hardware. Below is a screenshot of what gameplay may look like:



You must submit your C source code for plagiarism checking purposes. All submitted code will be compared through the use of plagiarism checking software. The penalty for plagiarism at QUT is severe and will be handled according to the QUT plagiarism policy. Extensions will only be granted for medical, work or exceptional family circumstances (as outlined in the relevant QUT policy). No other reasons will be considered.

Implementation Requirements

The implementation is broken down into a number of separate tiers. The first two of these are 1) full gameplay without opposition AI (artificial intelligence), and 2) AI on the opposing tank (worth a combined total weight of 60%). The remaining tiers (which comprise the remaining 40%) of the assignment will be released in part B at a later date.

The following detailed specifications outline the breakdown of marks. To receive the marks denoted for each section you must be able to perfectly demonstrate the exact requirements. The game must be able to be played by an unskilled user (your marking tutor). There is some expected variation in implementations but configurations (i.e. power range, gravity, etc.) should be tailored to make the game playable and winnable.

Individual version:

As inferred, the individual implementation must be completed individually. Any common code will constitute plagiarism and be dealt with accordingly. For this reason it is best to complete this version unless you are absolutely sure you want to do the group version (if you start on the group version, you can't both then use the code you started writing in a group).

1. Full gameplay without opposition AI (50%)

To complete this section, you should have a complete game that is playable from start to finish. The game should end when the opposing tank is hit. There should be the following:

- Start screen:
    - Display game title, student number, and name (1%)
    - Start the game by pressing a button (1%)
    - Countdown to game start (3 – 2 – 1 @ 1Hz) (1%)
- Game screen should have:
    - Two tanks with one on either side of the screen (tanks are 5 pixels wide and 3 pixels high with the top row only having the middle pixel black) (2%)
    - A flat terrain on which the tank sits. The terrain should cover all pixels from the 5th bottom row and below. The top boundary of the terrain is represented by a solid line, with the area below the boundary represented by alternating pixels (3%)
    - A visual representation of both the range and direction of the tank's current aim. A simple option would be a straight line from the top of the tank with varying angle and length (3%)
    - A bullet, 2x2 pixels (2%)
- Game physics:
    - The bullet trajectory must emulate realistic physics by:
        - Using a vertical gravity that drags the bullet towards the bottom of the screen (5%)
        - Following a parabolic path that directly corresponds to the provided power and angle (5%)
        - Drawing a bullet path that is as smooth as possible (i.e. correctly place the bullet based on accurate floating point arithmetic) (5%)
    - Bullets can exit the top of the screen and return without interrupting the correct trajectory (5%)
    - Bullets explode when hitting the terrain or a tank without affecting the drawn pixels (5%)
- Control input:
    - The shot angle should be adjustable within the range of $10°$ to $80°$ (upwards from the terrain and towards the opposing tank). The angle must be incremented by visible values only (i.e. 1 button press must correspond to a visible change in the tank's aim) (3%)
    - The shot power should be adjustable within the range of 10% to 100% of the maximum power value. Once again, the power must only be incremented by visible values (3%)
- Game control:

    - Left button (SW0) is used to adjust angle and power, and the right button (SW1) is used to set the value. Once set the shot should be fired (2%)
    - The aiming and firing process should be repeated until the opposing tank is hit (1%)
    - When the opposing tank is hit, a sequence of events (LED flashes and appropriate display on screen) should be performed before returning to the main screen (3%)

2. AI on the opposition tank (10%)

This section must be built on top of the previous section. Without demonstrating all of section 1 (i.e. have a complete working game), you cannot receive any marks for this section. Your implementation of this section should include:

- Control logic:
    - After the player's shot finishes the opposition tank should autonomously adjust the aim visualisation and then fire (2%)
    - The aim should be set to a **random** value (i.e. when restarting the game the opposing tank should select a different first aim) (1%)
    - After the first random shot, the AI should work through a process to refine its aim settings to get closer to hitting the enemy tank. More randomness must be incorporated into this refining process (4%)
- Game physics:
    - The bullet should correctly fire from the opposing tank in the opposite direction with all the same trajectory requirements as in tier 1 (3%)

Group version:

The group implementation requirements take into consideration the distinct advantage of having two contributors in development and debugging. For this reason, this version is considerably harder than the individual version.

To complete the group variant of the assignment you should make the following modifications to the hardware configuration:

- Both Teensy boards should be set up side by side on a breadboard
- The Teensy boards should use USART communications to synchronise the displays (this involves wiring and a correct hardware configuration procedure in your code)
- The game should stretch across both screens with the both screen's contents updating **at the same time**

Before you can receive any marks for the following implementations, the above must be working correctly!

1. Full multiplayer gameplay (50%)

To complete this section, you should have a complete game that is playable from start to finish. The game should end when either tank is hit. There should be the following:

- Start screen:
    - Display game title, student number, and name on each screen (1%)

  - A button should be pressed on each Teensy to start the game. When a button is pressed on 1 Teensy, the LEDs on the other Teensy must go on to show that the opponent is ready (1%)
  - Countdown to game start must be synchronised between both screens when both players have pressed the start button (3 – 2 – 1 @ 1Hz) (1%)
- Game screen should have:
  - Two tanks with one on the far side of each screen (tanks are 5 pixels wide and 3 pixels high with the top row only having the middle pixel black) (2%)
  - A flat terrain on which the tank sits. The terrain should cover all pixels from the 5th bottom row and below. The top boundary of the terrain is represented by a solid line, with the area below the boundary represented by alternating pixels (3%)
  - A visual representation of both the range and direction of the tank's current aim. A simple option would be a straight line from the top of the tank with varying angle and length (3%)
  - A bullet, 2x2 pixels (2%)
- Game physics:
  - The bullet trajectory must emulate realistic physics by:
    - Using a vertical gravity that drags the bullet towards the bottom of the screen (3%)
    - Following a parabolic path that directly corresponds to the provided power and angle (4%)
    - Drawing a bullet path that is as smooth as possible (i.e. correctly place the bullet based on accurate floating point arithmetic) (3%)
  - Bullets can exit the top of the screen and return without interrupting the correct trajectory (even if they exit the top of one screen and return on another) (5%)
  - Bullets when reaching the adjacent edge of each screen must then instantly appear on the other screen (**exactly** as though the two screens formed one large screen) (5%)
  - Bullets explode when hitting the terrain or a tank without affecting the drawn pixels (5%)
- Control input:
  - The shot angle should be adjustable within the range of $10°$ to $80°$ (upwards from the terrain and towards the opposing tank). The angle must be incremented by visible values only (i.e. 1 button press must correspond to a visible change in the tank's aim) (3%)
  - The shot power should be adjustable within the range of 10% to 100% of the maximum power value. Once again, the power must only be incremented by visible values (3%)
- Game control:
  - Left button (SW0) is used to adjust angle and power, and the right button (SW1) is used to set the value. Once set the shot should be fired (2%)
  - The aiming and firing process should be repeated until one of the tanks is hit (1%)
  - When a tank is hit, two different sequences of events (LED flashes and appropriate display on screen) should be performed on each screen (one for the winner, one for

the loser). These events must be synchronised in some manner (LED flashes, timing of text display, etc) before returning to the main screen (3%)

2. Multi- player with opposition AI – a third tank (10%)

This section must be built on top of the previous section (i.e. all of the multi-screen functionality must continue). Without demonstrating all of section 1 (i.e. have a complete working game), you cannot receive any marks for this section. Your implementation of this section should include:

- Control logic:
    - All three tanks should be randomly placed on either screen with appropriate firing angle limits. (2%)
    - LED0 on the corresponding player's Teensy should flash indicating the player's turn as the player's tank may be on the other screen (1%)
    - After the 2 human players shoot the opposition tank should autonomously adjust the aim visualisation and then fire (1%)
    - The aim should be set to a **random** value (i.e. when restarting the game the AI tank should select a different first aim) (1%)
    - After the first random shot, the AI should work through a process to refine its aim settings to get closer to hitting the enemy tank. More randomness must be incorporated into this refining process (3%)
    - If one human player's tank is eliminated, the order of firing turns should continue with the eliminated tank excluded (2%).