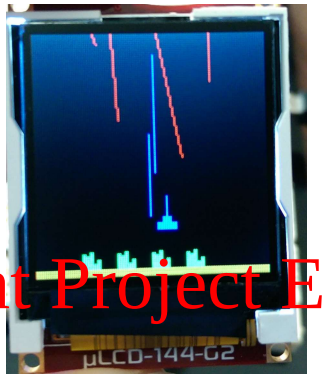
This project creates an Mbed version of Atari's classic Missile Command video arcade game that was wildly popular in the 1980's. In our Missile Command game, a city is being bombarded by missiles falling from the sky. A hovering aircraft (the player) flies just above the city and is able to shoot anti-missiles vertically to intercept the missiles threatening the city. If a missile gets past this guardian aircraft, it destroys a part of the city landscape. If all sections of the city are destroyed or if the aircraft itself is hit by a missile, it's Game Over! The objective of the game is for the player to fly the aircraft left and right across the city's landscape, intercepting as many missiles as possible for as long as possible. The more missiles are intercepted, the higher the score, and the player advances to more levels of the game. At each new level, the games gets more difficult: the missiles are generated at a higher rate and they fall at a faster speed.



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

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 missile generator, the city landscape display/undisplay maintainer, and aircraft display/undisplay features. You must complete the basic game functionality (described below) for 100 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 25 additional extra-credit points (5 additional features). So, to get a maximum of 125% on this project, your mbed program must support the basic game and 5 additional features. This document and the comments accompanying the shell code describe the features that you need to implement.

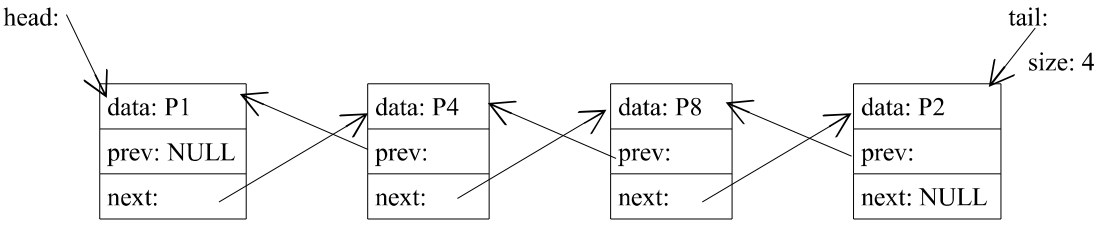
In testing and debugging your code, we strongly encourage you to set up a terminal interface to the mbed (see the section **USB Serial Debug** below) and to develop your code incrementally – making small modifications interleaved with testing to gradually implement features.

Basic Game Functionality

These features must be implemented and working correctly to earn the baseline 100 points:

1. Extend the partial Doubly Linked List implementation

The files `doubly_linked_list.h` and `doubly_linked_list.cpp` provide a shell for a utility module used by the missile command game. The functions in this module create and manipulate doubly linked lists of nodes. A doubly linked list is a linked list in which each node has both a next pointer and a previous (prev) pointer linking the node to its immediate neighbors. Here is an example:



The doubly linked list implementation needed for this project is minimal in that it supports only a few functions:

1. creating an empty doubly linked list,
2. creating a single node,
3. inserting a node at the head of the list (push onto the list),
4. deleting a node from the list (might be at head, middle, or tail)
5. destroying a list (freeing all nodes and their data and the struct holding head, tail, size),
6. accessing the list's size.

Some of these functions are already implemented in `doubly_linked_list.cpp` (items 1, 5, and 6). You need to complete those that are not yet implemented (items 2, 3, and 4). The file `doubly_linked_list.h` specifies what each of the functions is supposed to do. (Note that all the code for this project should be written in C, even though the file extension is "cpp".).

Assignment Project Exam Help

The shell code also provides a test function `testDLL` that is called by the main function which exercises the doubly linked list code you write for these functions. It generates print statements giving the status. If your code is incorrect for this part, it will generate an error code (e.g., -14) which can be looked up in `globals.h`. If you compile and run the shell code, you will see ERROR -14 show up on the uLCD display because of the missing functions you need to write.

https://powcoder.com

Add WeChat powcoder

2. Player aircraft 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 aircraft should move (fill this in `read_inputs` in `hardware.cpp`). Complete the move left and right functions in the `player` module. Your program should call them based on the accelerometer readings. It should keep track of where the aircraft is on the screen and it should never go off the screen. (The uLCD is 128x128 pixels.) Note: the terms "player," "aircraft," and "player aircraft" are used interchangeably in the code and comments.

3. Firing at missiles: When the player presses pushbutton 2, the aircraft should shoot an anti-missile vertically. Your program should display the vertical trajectory of each anti-missile as a blue line (as shown in the demo version) over time, eventually stopping at the top of the screen or when it intercepts a missile, at which point your program should make the blue line disappear. Your program needs to keep track of the position of the aircraft and of each anti-missile that is currently in flight (i.e., before it hits a missile or the top of the screen). Hint: the provide function `player_missile_draw` might be helpful. Note: the terms "player missile" and "anti-missile" are used interchangeably in the code and comments.

4. Detecting missile intercept and animating explosion: Your program needs to determine whether any of the anti-missiles that are currently traveling upward are within a certain radius of any incoming missile. (Initially, the radius should be 10 pixels, but make it adjustable so that you can set it larger or smaller depending on the current level of difficulty of the game.) If so, it

should draw an explosion as a simple circle to indicate the collision and then notify the missile generator that the missile has exploded (by marking its status as `MISSILE_EXPLODED`) which will cue the missile's deletion and erasure from the screen on the next `missile_generator` call.

5. Detecting and animating city destruction: Your program should determine when a missile has hit a city segment (hint: `city_get_info` may be useful). If so, it should draw an explosion as a simple circle to indicate destruction and notify the landscape maintainer to destroy and undisplay that city segment using `city_demolish`. The circle showing the city segment explosion should be a different color than the one used to show missile explosions.

6. Detecting Game Over: After all four city segments have been destroyed or if the player aircraft is hit by a missile (hint: you need to implement `was_player_hit`), the game ends with a "Game Over" message displayed on the screen.

7. Displaying score: display the number of missiles intercepted so far on the uLCD screen in the top right corner.

8. **Freeing up dynamically allocated memory**: The program should free up any dynamically allocated objects that are no longer being used. For example, it should free up all active missiles at the end of the game.

Assignment Project Exam Help

9. Advancing levels: After ten missiles have been intercepted while the game has been played at a certain level, it should advance to the next level. At each level, your program should set the rate at which the missiles are generated and the missile speed, increasing each at each advancement. It should also set the intercept radius to be smaller at each advancement. The current level number should be displayed on the uLCD in the top left corner.

https://powcoder.com

Add WeChat powcoder

10. Force level advance: When pushbuttons PB1 and PB3 are pressed simultaneously, the game should advance to the next level. (This will make it easier to test and grade the "Advancing levels" feature.)

Be sure to document your code. Points will be deducted for poorly documented code.

Extra Features (up to 25 points):

You can implement a subset of these features to earn an additional 5 points per feature, up to a maximum of 25 points:

More complex explosion animation – when a missile hits the city or is intercepted, more complex graphics can be used to animate the explosion (e.g., a series of concentric circles radiating out of the impact point).

Multiple lives – allow a player to earn multiple lives so that the game continues with a regeneration of the entire city landscape after the player survives for a certain amount of time.

Change aircraft appearance to something fancier such as a sprite (hint: `uLCD.BLIT` and other uLCD functions might be helpful – look up APIs and documentation on your uLCD on the mbed site under Hardware > Components).

Add MIRVs – these are multiple independently targetable reentry vehicles; a single missile splits into a small number of missiles that spray out in multiple directions. The altitude of MIRV deployment could decrease at higher levels of game play.

Enable diagonal anti-missile trajectories: use a pushbutton to toggle between two modes: one where the aircraft moves left or right (the baseline mode) and one where the aircraft *rotates* left or right when the board is tilted. When the aircraft has rotated and pushbutton PB2 is pressed and fired, the trajectory will be diagonal, rather than vertical. (Be sure to limit the rotation to the 180 degree arc above the aircraft so that the city does not come under friendly fire).

Smart/steerable anti-missile: allow player to steer the direction of the anti-missile while it's in flight using the accelerometer.

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) – 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/video (e.g., a triumphant one when you get a new high score).

Assignment Project Exam Help

Add sound effects – be creative. These could accompany explosions, firing, advancing to a new level, gaining a new life, hitting a new high score, etc.

https://powcoder.com

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/

Add WeChat powcoder

Use a pushbutton to create a new feature that can be used a limited number of times: for example, a super anti-missile with a larger radius of interest or that clears the screen, or speed boosters for anti-missiles or the aircraft.

To get more ideas, Google "Missile Command" to find playable versions of the original classic game from Atari (for example, https://my.ign.com/atari/missile-command). If the feature you would like to implement is not mentioned here, that's great, but be sure to check with an instructor or GTA to see if it would count.

USB Serial PC Communication: In testing and debugging your code, we strongly encourage you to set up a terminal interface to the mbed and to develop your code incrementally – making small modifications interleaved with testing to gradually implement features. You can set up a USB-based virtual serial port to display printf-style output from the Mbed on your computer. The tutorial can be found here: https://os.mbed.com/handbook/SerialPC. This includes a link (https://os.mbed.com/handbook/Terminals) detailing various options for setting up a terminal application on your PC. TeraTerm is a popular option for Windows (but use a more up-to-date download link, such as: https://tera-term.en.lo4d.com/windows).

The Serial pc object described in the tutorial is already set up for you in globals.h, so you won't need to declare it again. Any file that includes this header can print to the USB like so:

```
pc.printf("Hello, world!\r\n");
```