**ECE2036** **SPRING 2023**

# Lab 6 – Class Hierarchies and Pure Virtual Functions

*Assigned: April 10, 2023* *Due: April 25, 2023 by 11:59PM*

***FINAL EXAM REQUIREMENT:*** *You must turn in any parts you got from ECE before or during the final exam. Any parts you checked out of the library must go* ***back to the library*** *as well.*

**Part 1: (70 points)** In Lab4 you developed a technology to help protect humanity from a possible AI takeover!! In this lab I would like for you to use a class hierarchy for a very basic but soon to be classic video game called "AI Robot Takeover!" I have tried to simplify this game as much as possible so you can practice some of the core ideas discussed in class related to class hierarchies and polymorphism. These are some of the ideas that you will explore in this lab:

1. Class Hierarchy: You will make a hierarchy to implement your game's polymorphic screen manager. Your derived objects will call virtual functions like `draw()`, `move()`, and `erase()`. Your abstract base class should be `ScreenObject`, and your derived classes should be called `AIBot`, `BatteryChargers`, `ServerFarmers`, `CaringCapacitors`, `RebelResistors`, and `AIPoisonPill` that have specific implementations for `draw()` , `move()`, and `erase()`. In part 2 of the lab, you will add one additional capture target type or one additional poison type of your choosing to your hierarchy to demonstrate the concept of extensibility.
2. Polymorphic Data Structure: You are required to create an STL vector of `ScreenObject *` that will be used to manage your AI robot targets (either good or bad!). The new idea is that this is an array of BASE pointers that will point to various DERIVED class objects that will be created in the heap.

In this game, you will start with 4 AI target types and one bouncing AI poison pill moving around the screen, and you must maneuver your `AIBot` to capture these targets and avoid poison pills. You will have an `AIBot` avatar that you can move around the screen with the joystick component that you got working in Lab 3.

To give you an idea of what your game MIGHT look like, I have made a corny promo video to clarify and inspire your version of "AI Robot Takeover!" Take a look!

https://youtu.be/mPhRB9TmhF0

Also, I have given you some starting code to illustrate the use of the components on the mbed board in a tarball on the canvas assignment that you can import into your IDE environment for mbed. Please note you will need to make significant changes to this file (e.g. make a class hierarchy and using the ideas of polymorphism) to get full credit for your lab. THERE ARE "BAD" THINGS IN THIS FILE THAT YOU WILL NEED TO CHANGE BUT IT WILL COMPILE AND RUN ON YOUR SYSTEM!! You will notice that it does provide a sprite of a AI robot avatar that you can maneuver with the joystick to try to capture a square. It also shows you how to post a score at the bottom of the screen; however, it does not check to make sure the AI robot doesn't go off the screen or much else for that matter!!

I have added a repository in github with this starter program. The github is published at:

https://github.com/ProfDavisGitHub/Lab6Spring2023-CodeProvidedOriginal

From the File dropdown menu on Keil Studio, choose clone and put in the above URL. This should create a clone of the project into your account. If you are having trouble compiling, you can copy source files into one of the files that you setup for Lab3.

If this is not working, I have also given you a tarball of the files that you can upload to your Keil studio account. To expand the tar file you will need to use the following command.

tar -xvf Lab6Spring2023-CodeProvided.tar

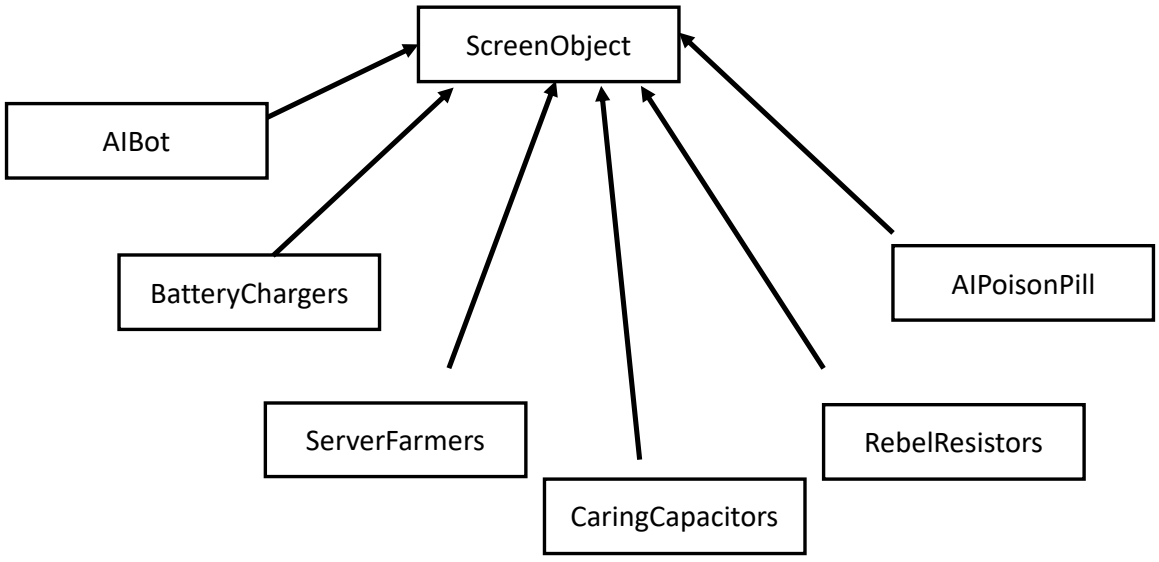
To make your AI robot move fast make sure that you set the highest baud rate on the LCD display to 300000 bits/sec; however, you must wait for about 0.3 seconds after you set the baud rate or else an error will occur. You should have something like the following at the beginning of your main.

```
uLCD.baudrate(300000);
wait(0.3);
```

Please note in general I want to give you freedom to make game design decisions that improve the basic game that you see in the demo video. However, there are some programming concepts that you must include in your code to receive full credit.

**Requirements Discussion**

1. I want you to have a class hierarchy that you implement for all your screen objects that looks something like following. If you want to add more levels to your hierarchy, please feel free to do so! Also, you should group any common data between all the derived classes and put in the ScreenObject class.



In addition, you need to have pure virtual functions of `draw()`, `move()`, and `erase()` that will be called from a vector of `ScreenObject` pointers. You should have some code that looks like this.

```
AIBot buzz1; // I will not manage in the vector
vector <ScreenObject *> AITargetOrPoison;
```

And you call the virtual functions like the following example for `draw()` in a polymorphic fashion.

```
//draw the AIBot targets or poison
buzz1.draw();
for(int i = 0; i < num_AITargets; i++)
      AITargetOrPoison[i]->draw();
```

Noticed I have kept my AIBot, which I called `buzz1` outside the vector of `ScreenObject` pointers (called `AITargetOrPoison`) that consistently has 4 targets and 1 poison pill. I made this choice because there are extra situations that you must deal with for the AIBot (especially checking to see if it intersects with `AITargetOrPoison` items).

3. Please have the target types move around the screen in some way. I have given you some ideas in the demo video. Your AIBot "captures" the target by simply having the AIBot touch the target (like in the demo code I provided for you). Once it is captured, the target will disappear briefly and the player will get 1 to 2 points for the capture. The target placement will then be randomized and placed at another location on the screen. However, if the AIBot eats the AI poison pill (in my game this is a big red bouncing ball adapted from Lab3), then the player loses 10 points. If the number of points goes below zero, then the game is over; however, if the player can get 50 points, the AIBot will be "full" and the player will win the game!!

4. In this lab, I would like for you to use the idea of a graphic "sprite." A sprite is simply a predefined set of pixel values for a small object that you would like to print to your screen and potentially move. Typically, you can move the sprite by erasing the old image of the sprint and drawing a new image of the sprite in a new location. I have defined the sprite for the AIBot in the sample code that I have provided for you. Please feel free to enhance your AIBot avatar. Further I would suggest that you put the sprite arrays ***in static memory***. Notice also that the sprite pixel color values are stored in a 1-D integer array. This integer array will be fed into a special member function from for your `uLCD` object that is called `BLIT`, which I will discuss later in this document.

```
#define SPRITE_HEIGHT 8
#define SPRITE_WIDTH 11

#define _ BLACK
#define X LGREY
#define R DGREY
#define Y RED

//sometimes putting in the global namespace and
//static memory can be a little more efficient for
//embedded programming

int AIBot_sprite[SPRITE_HEIGHT * SPRITE_WIDTH] = {
  _,_,_,_,X,X,X,R,R,_,_,
  _,X,X,X,X,X,X,X,R,X,_,
  X,X,R,X,X,Y,Y,X,X,X,X,
  X,X,X,R,X,Y,Y,X,X,X,_,
  X,R,X,X,X,X,X,X,R,R,R,
  _,_,_,R,X,_,R,R,_,_,_,
  _,_,R,R,_,_,_,R,R,_,_,
  R,R,_,_,_,R,_,_,_,R,R
};
```

To send this sprite image quickly to the LCD display, you can use the following command.

**uLCD.BLIT(x_pos, y_pos, SPRITE_WIDTH, SPRITE_HEIGHT, AIBot_sprite);**

5. Please use the joystick navigation component from Lab 3 to move your AIBot around the screen. You can see this operation in the bad sample code that I have given to you with the Nav_Switch class! The idea is that you have a while loop that is constantly checking the inputs of the joystick to update the position of your AIBot.

6. I would like for you to also use the push buttons for some fun and interesting options in your game. You will notice in the bad sample code that I provided to you that pressing the buttons changes the color of the square target. In your game be creative!

**Part 2: (20 points for extensibility!)** I would like for you to add an additional target type or poison type that you create on your own. Its shape can be of your choosing, but I would recommend that each has a height of 8 pixels and a width of 11 pixels to be consistent with other sprites. (Not a requirement.. just a recommendation.)You will have to make some changes to the main code as well…but not much! This is the power of polymorphism!

**Part 3: (10 points)** Using the speaker, add an intro jingle and an exit jingle to your game. You could even add some capture sound effects or sound effects when AI Robot poison is consumed. I will leave this to your discretion!

**EXTRA CREDIT:** (5 points) Wow your TA with your game and they can give you up 5 points extra credit!! The wow factor has to be well beyond the demo game that I have in the demo video!

## Turn in instructions

1. **Option 1**: Demo your lab to the TA during their lab hours. Print a copy of the checkoff sheet in Appendix A for the TA to sign. Make sure that you show them your code during the check-off so that they can see your class hierarchy. In person demos are the ONLY way to get extra credit for early completion.
2. **Option 2:** Post a video on YouTube showing you play the game to show off all its features! Please start the video showing your face and stating your full name. Also please step through your code to show your class hierarchy as well. You cannot get early turnin extra credit with this option; however, you are still eligible for the "WOW-THE-TA" extra credit!! Post the links to your YouTube video in your canvas submission.
3. For both options MAKE SURE THAT YOU POST a tarball of your source code on the canvas assignment along with a picture of your signed check-off sheet. The TAs will need to look at your code for reference. You can export the code in a zipped format from the mbed site like you did in Lab 4.

**APPENDIX A**
**Grade Check-off Sheet**

**Student: _____ Signature of TA: _____ Date: _____**

**Completed Part 1 Requirements:**
Show the TA your class hierarchy (30%) _____
Show the TA your working game (40%) _____
**Completed Part 2 Requirements** (20%): _____
**Completed Part 3 Requirements** (10%): _____

Additional Comments: _____

**Additional points can be *deducted* at the discretion of the TA according to :**

| Element | Percentage Deduction | Details |
|---|---|---|
| Used a class hierarchy improperly. | Up to 30% | This is a requirement for the program. |
| Code does not compile | Up to 30% | TA will try to give you some credit for non-functioning code |
| Game not even close to the DEMO game used as an illustration | Up to 30% | You have some flexibility in making a game that is unique to you and fun. But if the functionality is too basic the TAs may take off points. |
| Did not use Self-Documenting Coding Styles | 5%-10% | This can include no indentations, using unclear variable or class names, and/or unclear comments. |
| Did not post a zipped copy of the source code for this project on canvas. | 5% | Make sure you do this in the assignment portion of canvas. |

**TURN IN EARLY POLICY ** You only get this credit if you demo this in person with the TA**

| Element | Percentage Extra Credit | Details |
|---|---|---|
| Turn In April 18 (or before) | 4% | You must have complete check off with TA |
| Turn In April 19 | 3% | You must have complete check off with TA |
| Turn In April 20 | 2% | You must have complete check off with TA |
| Turn In April 21 | 1% | You must have complete check off with TA |

**LATE POLICY**

| Element | Percentage Deduction | Details |
|---|---|---|
| Each Additional Day | 15% per day | The weekend counts as one day. |

## APPENDIX B
## Mbed Graphics and Timer Library and Constants.

This is available online through mbed.org, but I thought I would include a few items that might be useful to your system design.

These are constants that define the colors that you can use in this project.

```
#define WHITE 0xFFFFFF
#define BLACK 0x000000
#define RED   0xFF0000
#define GREEN 0x00FF00
#define BLUE  0x0000FF
#define LGREY 0xBFBFBF
#define DGREY 0x5F5F5F
```

These are some of the member functions that you can use with your uLCD_4DGL objects that you instantiate.

```
void uLCD_4DGL :: pixel(int x, int y, int color)    // draw a pixel
void uLCD_4DGL :: line(int x1, int y1 , int x2, int y2, int color)
void uLCD_4DGL :: filled_circle(int x, int y , int radius, int color)
void uLCD_4DGL :: circle(int x, int y , int radius, int color)    // draw a circle in (x,y)
void uLCD_4DGL :: rectangle(int x1, int y1 , int x2, int y2, int color)    // draw a rectangle
void uLCD_4DGL :: filled_rectangle(int x1, int y1 , int x2, int y2, int color)    // draw a
rectangle
```

These are some of the Timer member functions that are available with the mBED system that are given as follows. You will have to make a Timer object to use these member functions.

```
void    start ()
        Start the timer.
void    stop ()
        Stop the timer.
void    reset ()
        Reset the timer to 0.
float   read ()
        Get the time passed in seconds.
int     read_ms ()
        Get the time passed in mili-seconds.
int     read_us ()
        Get the time passed in micro-seconds.
```