ECE 2036, Fall 2021
Lab 6 - Class Hierarchies and Pure Virtual Functions
Assigned: November 10th, 2021
Due: December 3rd, 2021

In this lab I would like for you to use a polymorphic screen manager for a very basic video game called "**World in the Balance.**" The year is 5242 and the information from Voyager spacecraft that was launch in 1977 has been discovered by the savage and alien Concal Empire. Their goal is to conquer the human race with their version of the ultimate weapon—the Death Star. Hey, it worked for Darth Vader! The only way to save planet earth and the human race is to harness the geothermal energy at the center of our planet and turn the whole planet into a maneuverable space ship so that we can find another star to orbit that is hidden from the Concal Empire. For your video game, I would like for you to make the first level only which is "Get through the Kuiper Belt." In short, the player will need to navigate "spaceship earth" through a dense asteroid field. You will use the accelerometer in mbed kit to control the movements of the earth between asteroids. Your artist design team has decided to use a retro asteroid look for this level that is illustrated on the next page.

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 classes will call pure virtual functions like draw() and update(). The class hierarchy is specified in the following pages.

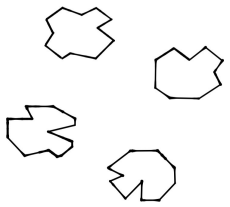
2. Polymorphic Data Structures: You will create an array of ScreenObject * that will be used to manage the asteroids in the Kuiper belt that spaceship earth must navigate. The new idea is that this is an array of BASE pointers that will point to various DERIVED class objects.

In this game, you will start with 4 different asteroids that will come in from all directions on the LCD screen. Your spaceship earth will move like the circle in Lab 3 with the accelerometer so that you can dodge the asteroids. Please note that as an asteroid goes off the screen you must have another asteroid enter the screen from a random location around the edge. Please watch the following video as a guide to design your own game. Please feel free to spruce it up beyond the basic required components.

https://youtu.be/I94lCZsMuvY

Requirements Discussion

1. Asteroid Design: I would like for you to have at least 4 asteroids flying through the screen at one time. Please match the below drawings the best you can. I have provided the first one for you in code on the next page. Also, if your earth collides with an asteroid, please create an appropriate explosion.



2. Asteroid Movements: Use the rand() function to pick the initial locations around the edge of the screen (~128 x 128 pixels) to introduce a new asteroid. Once an asteroid leaves the screen, create a new asteroid to come in from a different location. You should also randomly assign the slope of the trajectory of each asteroid. To do this, randomly assign a deltaX and deltaY that you maintain in the AsteroidAbstract class that is used to update the asteroid's position. Please see the video for an example of this.

3. Begin main() Function: Here is some code that you might need to start the game and get the LCD screen working correctly.

```
int main() {

    uLCD.baudrate(300000);
    wait(0.2);

    srand(time(0)); // do this srand call here ONLY... no where else in the code!

    ScreenObject * ActiveAsteroids[NUM_ASTEROIDS];

    SpaceShipEarth ship;

    //more code after this!
```

4. Game Timer: To complete the game, the user must dodge asteroids for 30 seconds. Please use the time function in the ctime library to keep track of the elapsed time. If the player can successfully dodge the asteroids for the entire time, then he or she has cleared this level. Yay! Otherwise, the earth explodes and the game ends. If the player survives the entire time, then clear the screen and print a congratulation to the user.

   I would like for you to have a TIMER BAR on one side of the screen that gets shorter as time progresses. In the video, which is hard to see, there is a blue bar at the top of the screen that shortens as time progresses.

```
time_t startTime;
startTime = time(0);
time_t timeElapsed = time(0)-startTime;
```

5. Graphic Sprites: In this lab, I would like for you to use the idea of a graphic "sprite." I have defined the sprite for ONE of your asteroids and a spaceshipEarth as seen below. You will use the mbed member function for the uLCD display called BLIT to display the sprite to the screen. Here are some code snippets to help you. You may define your sprites in the global namespace as seen below in order to maximize performance.

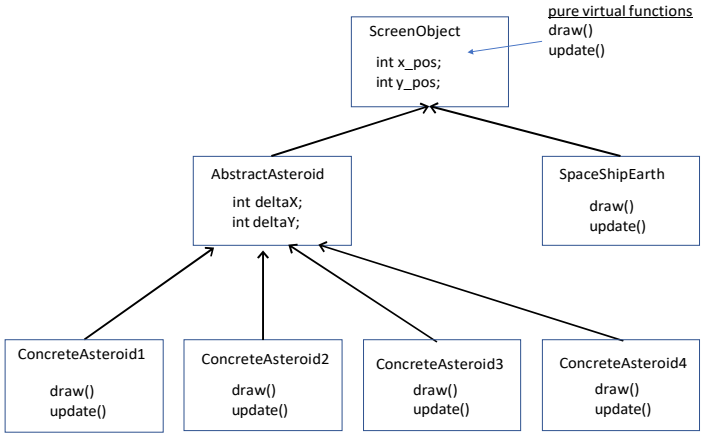
```
#define ASTEROID_HEIGHT 12
#define ASTEROID_WIDTH 15
#define SPRITE_MAX 15
#define EARTH_WIDTH 10
#define EARTH_HEIGHT 10
#define EXPLOSION1_WIDTH 20

#define SCREEN_MAX 125
#define SCREEN_MIN 1
#define NUM_ASTEROIDS 4

#define Q 0x808000 //OLIVE
#define I 0x008000 //GREEN
#define S 0xC0C0C0 //SILVER
#define C 0x17202A //UFO GLASS
#define D 0x797D7F //DARK GREY
#define L  0x00FF00  //LIME
#define P  0xFF00FF  //PINK
#define R 0xF1C40F //YELLOW
#define O 0xF39C12 //ORANGE
#define G 0xAAB7B8 //GREY
#define _ 0x000000 //BLACK
#define X 0xFFFFFF //WHITE
#define B 0x0000FF  //BLUE
#define r 0xFF0000  //RED

int asteroid_sprite_1[ASTEROID_HEIGHT * ASTEROID_WIDTH] = {
      _,_,_,_,X,X,X,X,X,X,X,X,_,_,_,
      _,_,_,X,_,_,_,_,_,_,_,_,X,_,_,
      _,_,X,_,_,_,_,_,_,_,_,_,X,_,_,
      _,X,_,_,_,_,_,_,_,_,_,_,X,_,_,
      X,X,X,X,_,_,_,_,_,_,_,_,_,X,_,
      _,_,_,X,_,_,_,_,_,_,_,_,_,X,_,
      _,_,X,_,_,_,_,_,_,_,_,_,_,X,_,
      _,X,_,_,_,_,_,X,_,_,_,_,_,X,_,
      X,_,_,_,_,_,X,X,_,_,_,_,X,_,_,
      _,X,_,_,_,X,_,X,_,_,_,_,X,_,_,
      _,_,X,_,X,_,_,X,_,_,_,X,_,_,_,
      _,_,_,X,_,_,_,X,X,X,X,_,_,_,_

};
```

```
int spaceship_earth1[EARTH_WIDTH *EARTH_HEIGHT] = {
    _,_,S,S,S,S,S,S,_,_,
    _,S,I,I,I,I,I,I,S,_,
    S,I,I,I,I,I,I,I,I,S,
    S,I,I,I,I,I,I,I,I,S,
    S,I,I,I,I,I,I,I,I,S,
    S,I,I,I,I,I,I,I,I,S,
    S,I,I,I,I,I,I,I,I,S,
    S,I,I,I,I,I,I,I,I,S,
    S,I,I,I,I,I,I,I,S,_,
    _,S,S,S,S,S,S,S,_,_,
};
```

To send this sprite image quickly to the LCD display, you can use the following command. The x_UPPERLEFT and y_UPPERLEFT specify the coordinates of the upper left corner of the sprite.

```
uLCD.BLIT(x_UPPERLEFT, y_UPPERLEFT, ASTEROID_WIDTH, ASTEROID_HEIGHT,
asteroid_sprite_1);
```

6. Class Hierarchy: I would like you to implement the following class hierarchy as a part of this lab. Make sure you use pure virtual functions where appropriate.



7. Collision Detection: You will need to determine when the asteroid and the spaceship overlap. To determine overlap please assume that the rectangle circumscribing the sprite determines overlap boundaries. To detect a collision, you should be able to check if each of the corners of one object are in the interior of a second object to detect a collision. In addition, I would like for you to create a polymorphic global function that determines if two objects overlap. The function signature might look something like the following.

```
bool overlap(ScreenObject & objectA, ScreenObject & objectB)
{
//you put stuff in here!
}
```

The key idea is that you will pass the objects as references to a ScreenObject as seen above. This function will return true if the two objects overlap and false otherwise.

Use the speaker and add some sound effects for your game. At the very least make a good explosion noise when the earth collides with an asteroid!

Turn in instructions

1. Demo your lab to a TA during his/her office hours. During the demo show them your polymorphic data structure that you used to store different asteroid objects.

2. Upload your zipped source code on the canvas assignment. The TAs or the professor will need to look at your code to make sure that you used a hierarchy. You can put all of your code in one file if you choose to do so.

APPENDIX A

Grade Sheet Checkoff

Have a class hierarchy and polymorphism in your solution (40%)

Illustrate correct operation to your TA (40%)

Have a working time counter bar on the side of the screen (10%)

Have sound effects at some point in your program (10%)

Additional points can be deducted at your TA's discretion according to the following criteria.

Element	Percentage	Details
Used a class hierarchy improperly	40%	This is a requirement for the program.
Does not compile	30%	Please make sure your code compiles.
Does not use self-documenting coding styles	5% - 15%	This can include using incorrect indentation, unclear variable names, unclear comments, or compiling with warnings.

Turn-in Policy

Element	Percentage	Details
Turn In before or on Nov 21	"+3%"	You must have complete submission.
Turn in between Nov 22 and 23	"+2%"	You must have complete submission.
Turn in between Nov 24 and 30	"+1%"	You must have complete submission.
Turn in between Dec 1 and 3	0%	No late penalty
Each additional day after Dec 3	"-20%" per day	The weekend counts as one day (i.e., Turning in on Monday results in a 20% reduction).