**OOP – Program 8 (Final Project)**

**Space Invaders**

**Fall, 2013**

**Due by 5:00 PM on Fri, Dec 6th**

**No Late Work Accepted**

**40 pts**

**You may pair up with one other person on this project.**

**Set-up:**

If you have not already done so, you must install the XNA framework as described on bboard. You must use the XNA framework to implement your program.

You are to implement a "space invaders" game. I am less concerned about the exact functionality and appearance of the game than I am that you exercise appropriate design and implementation skills.

Specifically, your solution must include the following:

* Appropriate use of multiple classes using composition and/or inheritance where appropriate. Do not force inheritance; use it only if it makes sense.
* Appropriate use of member functions and data.
* Appropriate distribution of data/functionality to classes. Each class should be defined in a separate source file (use Project->Add Class to add each class).
* You should use the C# List class where appropriate.
* Appropriate design and comments in the code

**Functional Requirements**

* Your game must begin with at least 10 aliens (a single row of aliens is sufficient).
* The game must have a 'Start' button that begins the action.
* The player moves the ship (horizontally only) using left/right arrow keys. The space key is the 'fire' button.
* You should be able to fire your ship’s gun such that you have multiple missiles en- route simultaneously.
* Periodically, a randomly selected alien should drop a bomb. The interval at which bombs drop should vary between a minimum of one second and a maximum interval (you can decide on the maximum interval). Multiple bombs may be en route simultaneously.
* The row of aliens should move down the screen toward the ship (you can decide the speed). A direct line of descent is satisfactory (i.e., aliens don't have to wiggle down the screen). If any alien reaches the level of the ship, count this as a destroyed ship and restart the current level. You should establish a limited number of lives per level.
* Hits on aliens and the ship should be detected. An alien simply disappears when it is hit (of course, the missile disappears also). When the ship is hit, the number of lives counter should be decremented, and the state of the game is reset to the beginning of the current level (you can decide where the counter begins). If the number of lives is exhausted, the game is over. At this point, the Start button should be enabled so that the user can restart the game.
* If the player destroys all of the aliens on the initial level, you should re-populate with a different kind of alien. Level 2 aliens should appear differently than level 1 aliens, and should drop bombs differently than level 1 aliens (e.g., you could make the bombs bigger or faster or more frequent than Level 1 aliens). You are only required to have two levels. If the user kills all Level 2 aliens, the user wins the game. An appropriate message box should announce the win. When this box is clicked, the state of the game should be re-set to the beginning (ready for the Start button to be clicked).
* Labels on the form should be used to display the following information:
  + Number of lives left for the gunner.
  + Number of destroyed aliens.
* Your game should have a button labeled Settings. Clicking this button should display a dialog box from which the user can establish several game parameters, as described below. Your dialog must include at least the following controls:
  + An OK button
  + A cancel button
  + A combo list box for selecting skill level. From this combo box, the user should be able to pick either the Beginner, Intermediate, or Advanced level of game play. You can decide what each level means (for example, higher skill levels might move Aliens faster, or drop bombs more frequently, etc.). This level should default to Beginner.
  + A group box containing radio buttons. The radio button options should be “No cheat” and “Invincible”. “No Cheat” should be the default. If the user chooses Invincible, then Alien bombs do not destroy the ship.
  + You may add any other game parameters you wish; but all should have default values.
  + Any time the user changes settings, the settings should be silently stored in a text file. When the program begins, settings should be loaded from this same text file. If the file doesn’t exist or errors occur during file I/O, these errors should be silently ignored.
  + Since each game parameter has a default value, the user should not be forced to use this dialog before they can play the game.

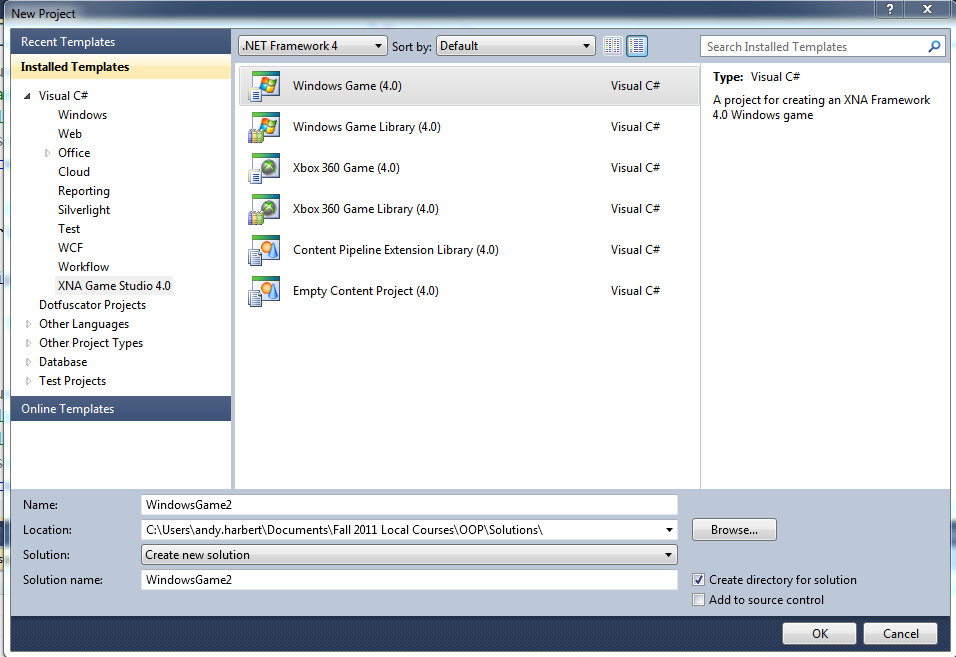
**How-To’s and Hints**

1. **Sample Project**

Download and unzip the xnaDemo project on bboard (under this assignment link). This illustrates all of the mechanics you'll need to implement the game/drawing part of this application.

1. **Creating a new Project**

From Visual Studio, select **File->New-Project**. Under the **Visual C# template** (see left hand side of screen), select **XNA Game Studio 4.0**. In the middle pane, select the project type **Windows Game (4.0)**. At the bottom, give the project a name, etc., and click OK.



1. **Main Flow of the Application**

Once the game play begins, your primary control will be accomplished by several Game1 member functions. These functions should not contain all of the processing logic, but the application’s behavior will be ***initiated*** from one of these functions. Skeletons for these functions are all generated when you create an XNA project.

|  |  |
| --- | --- |
| Controlling Functions | |
| Function | Function’s Primary Responsibilities |
| Initialize | This is where one-time initialization happens (just like a constructor) |
| LoadContent | This is where you load textures (pictures) needed for your game. |
| Update | XNA calls this function approximately 60 times a second. This function is responsible for updating the positions of various objects (moving Aliens, bombs, and missiles). It also determines when something is hit or a projectile has moved off the screen. |
| Draw | This function gets called *after* every call to Update. This function doesn’t change the state of any data structure, it simply draws the representation of the game on the screen for the current frame. |

1. **Update**

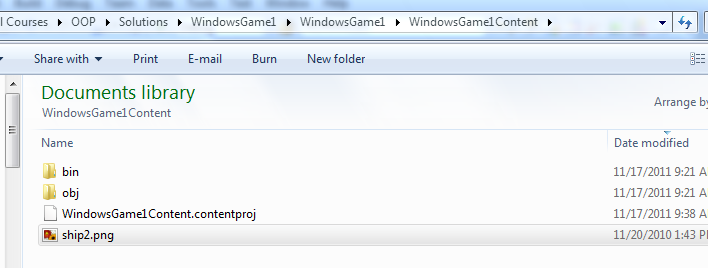
As mentioned above, this is where most of the action happens. On every call to this function, you will need to perform the following major activities:

* Update the position of all *active* missiles/bombs. Take appropriate action when the missiles/bombs hit something, or remove the bomb when it passes from the field of play. Each bomb should be represented by an instance of a class (maybe a "Bomb" class). A Bomb should know where it is, and should be able to update its position. It should also have a *getter* property that returns a Rectangle. You will need to know the bounding Rectangle so that you can perform collision detection with the ship. Bombs should also be able to draw themselves (given appropriate parameter). A List of Bombs should be used to represent all Bomb instances.
* Determine the frequency with which the alien invaders drop a new bomb. Since we’d like to give the hapless humans a chance, it is unreasonable to drop a new bomb every time Update is called, so you will want to establish a counter that you can use to determine how frequently to drop a new bomb. Notice: multiple bombs may be in play simultaneously (hence the need for a List of Bombs).
* Determine whether the Bomb has hit the ship, or simply run off the screen.
* Determine whether the left/right arrow key is down, and reposition the ship appropriately.
* Determine whether the space key has been pressed, and create a new missile. (NOTE: firing a missile should require the user to press and release the space key…this is different than the ship movement keys that may be held down continually).
* Your application should maintain a List of missiles, handled in a similar manner to Bombs.

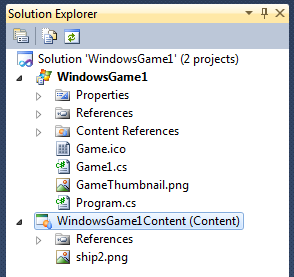
1. **Making the Panel of Controls**

Since this is an XNA project, we do not have the luxury of dragging controls onto a "Form1". However, this is a good place for us to see how GUI's can be created programmatically (see the sample project for an example).

1. **Drawing Aliens, etc.**
   * Use a bitmap editor (e.g., Paint) to create bitmaps of the images to use for the aliens, the human ship, missiles, and bombs. In the bitmaps, make the background black so that it will blend in with the black space background (or, you can use an editor that allows you to create transparent pixels). Don't spend much time on this! You don't get extra points for pretty artwork; stick figure aliens are fine. You can go back and pretty things up once you get all of your requirements satisfied.
   * To make the bitmap file accessible to your program, you need to first copy the file to the “Content” folder that Visual Studio created when you created the project (the default name for this folder is WindowsGame1Content). Notice that this folder is two levels below your top-level solution folder. In the example below, I have put the ship2.png file in this folder.



* + Next, within Visual Studio, click RIGHT on the WindowsGame1Content (Content) Project. Select Add->Existing Item, and navigate to the file you previously copied to the Content folder. Double click on the file to add it to the project. After this is complete, the file should appear under the Contents project, as illustrated below:



Your code will reference this bitmap with a statement such as the following (in LoadContent()):

ship = Content.Load<Texture2D>("ship2");

Further examples are provided in the demo project.

1. **Class Design**

* The functions in the Game1 class should *initiate* things, but not DO everything. For example[[1]](#footnote-1), you should have a Bomb class. An instance of this class represents a live Bomb (as described earlier). A Bomb should be able to draw itself (the draw function must accept a spriteBatch parameter passed to it from the Draw function).
* The list of Bombs should be in a containing class, say, AlienArsenal. AlienArsenal should have an updateStatus function that the Update function can call. updateStatus would be responsible for accessing every Bomb, updating its position, and if necessary remove it from the list of bombs (e.g., because it hit something or ran off of the screen). AlienArsenal would also have a draw() function called from the Game1.Draw function, but AlienArsenal.draw() would simply be a wrapper function that works through the Bomb list, telling every Bomb to draw itself.
* Use aggregation appropriately. For example, Game1’s private data should include an instance of AlienArsenal, but Game1 would not directly contain an instance of Bomb.
* The whole point of having classes is to be able to group functionality with data in order to modularize your application, so you should aim towards this ideal in the design of your application. Don’t lump everything into the Game1 class!
* *50% of your grade for this program is based on your design, efficiency, and maintainability issues. The other 50% is based on correct implementation of the functional requirements.*

**Turn-In Instructions**

* *You must include a Word file named DOCUMENTATION, and zip it up with your project folder. This Word file should contain two items:* 
  + *Instructions for running the game*
  + *List of incomplete functionality or known problems/bugs. I will penalize less for known problems than for problems I discover during testing that you did not document.*
* Make sure that when you zip your project folder, you are zipping the ***entire folder*** (this includes the solution/project file, the source code, the resource files, bitmaps, etc.). Make sure that all referenced items are included in your zip file (YES, YOU SHOULD TEST EVERYTHING BEFORE SUBMITTING IT). You should treat this like a real-world project; if you have to resubmit then you will receive a 10% penalty).

1. This is only an example. It is not intended to be an exhaustive nor exclusive list of functionality for various classes. [↑](#footnote-ref-1)