CS 3201 – Fall 2024
Assignment 3, 20 pts.
Due: Wed, Oct 23rd at 9a

Objectives

- Add new functionality to existing code.
- Design basic algorithms.
- Create new classes by taking advantage of inheritance to limit the amount of new code necessary and leverage polymorphism to reduce code duplicity.
- Incorporate abstract classes and methods to create a more realistic hierarchical class structure and factor out common attributes and behavior.

Notes

- Please read through all the requirements before you begin, so you know what the application is supposed to do when you are done with this iteration of the assignment.
- You must use ReSharper to verify your code meets the style requirements.
  - Make sure you are using the class ReSharper configuration file.
  - Use ReSharper to both clean up and inspect the code.
- **Any program that does not compile will receive a zero (0). Partial credit is not possible for any program that does not compile.**

Specifications

Getting started

1. The goal of this assignment is to create a first iteration of a simplified Galaga clone (https://www.free80sarcade.com/galaga.php). The starting point of this project is the starter project provided in Moodle. Download the project and Step 1 is the following:

   a. Change the text in the title bar of the application to display the following:

      Galaga by *Lastname* A3

2. Iteration 1 functionality for the galaga clone will introduce the following functionality.

   a. Keep the player ship from moving outside the bounds of the background canvas.

   b. Introduce three (3) level of enemies that will be placed and move back and forth during game play.

   c. Have the player be able to shoot at and destroy enemy ships.

   d. Have the level 3 enemy shoot at and destroy the player.

   e. Add a scoring mechanism.

   f. Incorporate a game over functionality when the player destroys all the enemy ships or the player is destroyed.

3. Add version control to your project:

    a. In Visual Studio, add git-based source control to your project. This can be down by selecting Create Git repository… in the context menu for the solution.

        i. Make sure to push to GitHub as a new remote repository.

        ii. It must be a **private** repository.

        iii. Name the repository *FirstnameLastname*Galaga.

    b. Share the GitHub repository with (yoder531) dyoder@westga.edu. This can be done in the Settings → Collaborators section of the repo within GitHub.

    c. Use version control as you develop your project to commit changes as you develop your code. Make sure you commit your changes locally and also push them to the remote repository.

Requirement #1: Keep player ship within the bounds of the background

1. Currently, the player ship moves back and forth horizontally with the left and right arrow keys. However, the player ship can move beyond the bounds of the black background canvas. Add functionality so that when the player ship encounters the left or right canvas boundary it will prevent additional movement in that direction so that the player ship stays within the bounds of the game background.

    a. You may change the look and size of the player ship if you like (not required), but if you do, please do not make the player ship too large.

Requirement #2: Add enemy ships

1. Add three different types (levels) of enemy ships. The requirements for the enemy ships are as follows:

    a. Type 1/level 1 will be the bottom row of ships and will be the most basic enemy ship.

    b. Type 2/level 2 will be the middle row of ships.

        i. Level 2 ships **must** be based on the level 1 ships. In other words, it will be a level 1 ship with additional feature(s) added. You only need to add one additional feature, but can add more if you like.

    c. Type 3/level 3 will be the top row of ships. Level 3 will be the most advanced enemy ships for iteration 1 of the game.

        i. The level 3 ship must be a completely new ship.

    Note: The enemy ships do not need to be the same size as the player ship, but they need to be a reasonable size in relation to the player ship.

2. The enemy ships must be placed in three rows of 2, 3, and 4 enemies from level 1 to 3, respectively, for a total of 9 enemies. Do not add more than this number of enemies in this iteration of the game.

    a. The initial placement of each row of enemy ships must be **centered horizontally**.

    b. Do not hard code the exact placement of each individual ship. You may specify the y location for each row, but the code to place each ship in the row should be done programmatically. Note: in future iterations each row will have more ships.

3. The enemy ships need to move back and forth horizontally as the game is played.

    a. The movement can be implemented in **only one** of the following two ways:

        i. They need to move at least 5 steps in each direction from its initial placement. In other words, the initial place is the center of its movement. For example, if the enemy ship initially moved to the right 5 steps. It would then move back 10 steps to the left, and then 10 steps back to the right, etc.

        OR

        ii. Using the initial placement as the boundary of the movement. The movement would then be to move 10 steps in one direction and 10 steps back so that it returns to its starting point.

    b. The movement of each ship will continue throughout the game play.

Requirement #3: Player firing capability

1. Add functionality so that the player can shoot bullets at the enemy ships via the space bar.

    a. When a bullet is fired it will continue on the vertical path it started on.

2. If a bullet hits an enemy, the enemy should be destroyed (removed) and the score updated. (See Requirement #5 for scoring.)

    a. If a bullet hits an enemy, the bullet is destroyed also. In other words, the bullet cannot continue on through the enemy and destroy more enemies.

    b. All collision detection for the game can be done by using a bounding box.

3. The player should not be able to have more than one bullet active at a time. In other words, if a bullet is still visible on the screen the player cannot fire another bullet.

Requirement #4: Enemy firing capability

1. Add functionality so that the level 3 enemy can fire bullets at the player ship.

    a. When a bullet is fired it will continue on the vertical path it started on.

   b. The bullet should be fired at a random interval by the enemy. In other words, there should not be a predictable pattern as to when an enemy will shoot.

Requirement #5: Add a scoring mechanism

1. Add functionality to display the player's score.

2. Different points should be earned for level 1, 2, and 3 ships.

3. Update the score as enemies are destroyed.

Requirement #6: Game over mechanism

1. Add functionality so that if the player destroys all the enemy ships it informs the user that they won.

   a. For the first iteration, there will only be one round/level to the game.

2. Add functionality so that if the player dies, Game Over is displayed.

   a. For the first iteration, the player will only have one life.

Requirement #7: GitHub and version control requirements

1. Commits to your repository must have appropriate comments for the commit.

   a. The commits should be for functionality that is completed, not just small textual or formatting changes.

2. If there are in bugs or missing functionality, then within GitHub's *Issue Tracking* create an issue for each item.

3. Once you have completed the assignment tag the completed assignment changeset with the following label Assignment3Submission.

   a. You can create the tag in Visual Studio, but if you do so you will also need to push the tag from Visual Studio.

   b. You can also create the tag in GitHub by creating a release.

Implementation notes

The implementation needs to follow best practices in application design and coding practices.

The class design for the player already employs model-view separation.

Submission

Make sure you clean your solution (Build → Clean Solution) and then zip up your project into a ZIP file called *FirstnameLastname*A3.zip and submit the assignment by the due date. Note: I will clone the project from GitHub, but I want to make sure I have an exported version of the project.

Grading rubric

- *Any program that does not compile will receive a 0. Partial credit is not possible for any program that does not compile.*
- *If a program is only partially complete and a category cannot be accurately assessed you will not receive full credit for that category.*
- *One point will be deducted from your grade, if your submission does not follow the naming convention.*

Grading breakdown

	Exceptional	**Acceptable**	**Amateur**	**Unsatisfactory**
Functionality	12 pts. – graded on a continuous scale.			
Readability	2 pts.	1 pt.	NA	0 pts.
Implementation	4 pts.	3 pts.	2 pt.	1 pts.
Documentation	2 pts.	NA	1 pt.	0 pts.

Grading description

	Exceptional	**Acceptable**	**Amateur**	**Unsatisfactory**
Readability	The program is exceptionally well organized, very easy to follow, and there are not any ReSharper warnings.	The organization and readability of the code could be improved and/or there are a few ReSharper warnings that were not legitimately explained as to why they still exist.		The code is poorly organized and very difficult to read and/or has lots of ReSharper warnings.
Implementation/ Reusability	The implementation follows best practices. The code is implemented and organized so it can be maintained and reused.	The implementation mostly follows best practices. Most of the code could be maintained or reused. The implementation and organization are useable but could be improved.	The implementation does not follow many of the best practices. There are several issues with the implementation and/or organization of the code.	The implementation is not following best practices. The code is not implemented nor organized for maintainability and reusability.

Documentation	The required documentation is complete, well written and clearly explains what the code is accomplishing. No redundant inline commenting.		The required documentation is missing in some places and/or is not useful or redundant in nature.	The documentation is poor and/or very incomplete.