CS 3201 – Fall 2024
Assignment 5, 30 pts.
Due: Friday, Dec 6th at 9a

Objectives

- Add new functionality to existing code.
- Determine an approach to solve a problem and implement that approach in code.
- Demonstrate an understanding of a class hierarchy by determining the best place for state and behavior that eliminates code duplicity and promotes reuse.
- Refactor existing code to reduce code smell and employ good class design principles.
- Apply design patterns as necessary.
- Apply clean coding practices.
- Use a version control system to share and work on a software project with other developers.
- Use issue tracking and a Wiki to help manage a software project.

Notes

- This assignment will be a team project. The teams for this assignment are posted in Moodle. The specifications of how the team will interact and work on this assignment are also included below.
- 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.
- Make sure you use ReSharper to verify your code meets the style requirements.
  - 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.**

Requirements

Getting started #1: Repository

1. Select one of your team member projects as a starter for this project and do the following:

   a. Rename the project and the repository: Team*X*Galaga, where *X* is your team letter. It must be a **private** repository.

   a. Make sure each team member has access to the repository.

   b. Add a wiki to the repository.

   c. If you haven't done so already, share the GitHub repository with (yoder531) dyoder@westga.edu.

Getting started #2: Renaming application

1. Change the title bar of the application to: Galaga by *List the last name of each team member*

Getting started #3: Refactor and debug

1. Refactor the Assignment 4 submission to eliminate all class design and code smell items that have been discussed in class and noted in any specific feedback.

   Items that need to be addressed include, but are not limited to, the following:

   - Placing functionality and corresponding state information in appropriate model and view classes.
   - Eliminating duplicate code within a class by leveraging private helper methods.
   - Eliminate data duplicity within a class hierarchy.
   - Eliminate data duplicity between collaborating classes.
   - Eliminating duplicate code within a class hierarchy by placing functionality in the appropriate base class.
   - Address readability, e.g.,
     - Appropriate use of enums and/or constants for magic numbers
     - Formatting and ReSharper issues
     - Appropriate number of levels of code within a method
     - Writing self-documenting code
     - Adding required and appropriate documentation

2. Remove all bugs from the assignment, so that the assignment works according to the Assignment 4 requirements.

   a. Note: If the project had errors or missing functionality, those issues must be addressed in this submission.

Functionality requirements

The functionality will be divided into separate parts. Each part is worth a specified number of points. Your team is not required to complete Parts 2 and 3. However, the total points earned will be affected if you do not. Please note: Part 3 will not be graded, if you do not successfully complete **all** of Parts 1 and 2.

The requirements of each part are given below. Do not move onto the next part until you have successfully completed the previous part.

Functionality – Part 1 – 7 pts.

Change the size of the canvas

1. Change the size of the game canvas to make it a different width and height than it currently is.

Ship factory

1. Create a ship factory that will create and return a ship anytime one needs created in the game.

Sound effects

1. Add sounds effects to the game for the following:

    a. Player ship firing bullet
    b. Player ship being destroyed
    c. Enemy ship firing
    d. Enemy ship being destroyed
    e. Game over

    Note: The same fire and destruction sound cannot be used for the player and enemy ships.

Multiple levels

1. Add functionality so that there are 3 levels to the game.

    a. If you complete a level and still have lives remaining you will move onto the next level.

    b. If you complete all 3 levels, then the game is over.

    c. Make each level a little different and more difficult than the previous level. The game must be winnable.

Bonus enemy ship

1. Add functionality so that a bonus ship appears randomly at some point during the play of a level. The bonus ship can move horizontally across screen or it could fly in an attacking pattern at the player.

2. If the player destroys the bonus ship then the player will earn an extra life.

3. A sound should play when the bonus enemy ship is active.

4. The bonus enemy ship should fire as it moves.

Functionality – Part 2 – 6 pts.

Sortable high score board

1. Add functionality so that a top ten high score board can be displayed. The user should be able to view this board when the game is over.

2. When the game is over the player should be able to enter their name if they make the top 10 scores.

3. The high score board needs to be persistent through various invocations of the program.

4. The high score board should display the player's name, score, and the level completed.

    a. The default ordering should be by the player's score/name/level, but the high score board should be sortable by the following multi-level sorts:

        i. Player score/name/level
        ii. Player name/score/level
        iii. Level/score/name

5. The display of the high score board must use MVVM.

Power up

1. Add functionality so that if the player destroys the bonus ship in addition to an extra life, the player will obtain some type of temporary power up.

2. When the power up is active there should be some indication that it is active, either via a display or sound.

Different enemy ship movement patterns

1. Modify the functionality so that when moving to a new level of the game that each row of ships moves differently. For example, the first level of the game all ships could move in the same pattern they have been moving. When reaching the second level of the game, level 1 and 2 ships could move in the same pattern, but level 3 and 4 ships move in a different pattern. Then for the last level of the game each level of ship moves in a different pattern.

Functionality – Part 3 – 3 pts.

For part 3 you only need to implement, three of the five items below. You can choose which three. You will earn one extra credit point for each additional one you implement.

Add a start screen

1. Add a start screen that allows the user to either start the game, display and view the high scoreboard, and reset the high scoreboard. If the high score board is displayed, there needs to be functionality to return to the start screen.

Shoot at the player

1. Change the functionality that for the level 4 enemies and the bonus ship that they fire directly at the player ship instead of shooting straight down. Once the direction of the bullet is determined it will stay on the same trajectory even if the player ship has moved.

Double player ship

1. Modify the destroying of the bonus ship so that on Level 2 and 3 if earned, instead of earning an extra life the player the player becomes two ships wide with each ship firing. Each ship can then be destroyed individually. The temporary power up should still apply to both player ships.

Explosion animation when player or enemy dies

1. Add an explosion animation of at least 3 frames when the player or an enemy are destroyed.

Attacking patterns

1. Add functionality so that the level 4 ships will fly in an attacking pattern at the player ship at some random interval. An enemy ship should be able to fire while it is in its attacking pattern. During the attacking pattern the enemy ship should fire at a faster rate.

WOW factor – 2 pts.

1. WOW factor. Add some additional functionality to your project that was not specified. Please note your WOW factor feature(s) in the project wiki. The amount of points earned for the WOW factor will be based on how much of a wow it is.

   The WOW factor will only be graded if at a minimum Parts 1 and 2 are completed.

Github and version control requirements

1. Each team member must make at least five commits.

2. Within the repository there must be at least two "merges" within the repository. If you are on a team of three, there needs to be at least four merges where each team member has code that is involved in the merge.

3. In the wiki do the following (each team member must contribute to the wiki):

   a. Provide indicate what parts and individual items of the project your team completed.

   b. Provide a screen capture of your game.

   c. Provide a description of any WOW factor item(s) or note that the WOW factor was not attempted.

   d. Make sure your team uses appropriate headings within the wiki to offset the required sections.

4. In *Issue Tracking* do the following:

   a. Setup issues for any bugs and clean code refactoring that need to be addressed within the "starter" project.

   b. Setup issues for each of the additional requirements for the project.

   c. As a team assign specific issues to each team member.

   d. Make sure each team member resolves the issues within Issue Tracking as they are fixed.

   e. Any outstanding issues or missing functionality must be noted as unresolved issues within Issue Tracking.

5. Once you have completed the assignment tag the completed assignment changeset with the following label Assignment5Submission.

Moodle submission requirements

In Moodle each team member needs to do the following:

1. Specify an estimate of the percentage of work each team member did, e.g., John – 40%, Sally – 60%

2. State what you specifically implemented in the project.

3. Give a statement or two summarizing your team member's contribution.

4. Assign yourself and your team member(s) a grade on a 0-100 scale.

   Note: Your team member(s) will not see any feedback that you provide in Moodle.

Have the team lead export the tagged version and submit your completed project to Moodle by the due date. Verify the name of the exported file is: Team*X*Galaga.zip. Note: I will actually 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 – Total 30 points

- 16 pts. – Functionality
- 2 pts. – WOW factor
- 7 pts. – Implementation
- 3 pts. – Use of version control, wiki and issue tracking.
- 2 pts. – Team contribution based on commits in GitHub and evaluation from team member(s).
  - **If individually you do not contribute any work on this project, you will receive a 0 on this assignment regardless of what the rest of your team receives.**

	Exceptional	**Acceptable**	**Amateur**	**Unsatisfactory**
Functionality	16 pts.			
Readability	2 pts.	1 pt.	NA	0 pts.
Implementation	4 pts.	3 pts.	2 pt.	1 pts.
Documentation	1 pt.	NA	NA	0 pts.

Functionality

- Part 1 – 7 pts.
- Part 2 – 6 pts.
- Part 3 – 3 pts.

Implementation rubric

If a program is only partially complete and a category cannot be accurately assessed, you will not receive full credit for that category.

	Exceptional	Acceptable	Amateur	Unsatisfactory
Readability	2 pts.	1 pt.	NA	0 pts.
Implementation	4 pts.	3 pts.	2 pt.	1 pts.
Documentation	1 pt.	NA	NA	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.