Answer each of the questions below, demonstrating a full understanding of the concepts involved in each question. While you may discuss the questions with your classmates, your responses must be in your own words and your designs must be your own.

A zipped version of the executable of the video game “Star Castle” is available on the course web site at:


(It was the first assignment in last year’s Game Design, Development, and Technology course.)

This game consists of a keyboard-controlled spaceship, a pulsating space cannon in the center of the display, three concentric energy rings that spin around the cannon, and various mines that emerge from the cannon and, after traversing the energy rings one at a time, chase the spaceship.

Using the keyboard arrow keys, the user may rotate and propel the spaceship around the display (with wraparound). The spaceship is never allowed to enter the boundary formed by the outermost energy ring, bouncing instead off of its surface when such a collision occurs. Using the spacebar, the user may launch missiles from the front of the spaceship. The missiles are used to damage and destroy the linear segments that comprise the energy rings, with one hit causing a segment to glow brighter and a second hit causing it to explode. Exploding all segments in a given ring causes a new ring to be generated. A direct hit from a missile can also destroy a mine.

When an entire energy ring is destroyed, it automatically regenerates. However, if the user fires a missile down a path through partially destroyed rings, the missile may strike and destroy the cannon, causing the rings and cannon to implode and then explode before regenerating.

Note that the cannon always rotates so that it will point at the spaceship’s current position. Exploding the ring segments in a manner that affords a shot at the space cannon will also give the cannon a shot at the spaceship. The cannon fires plasma balls that destroy the ship upon contact.

Protective mines are generated by the cannon and placed on the interior energy ring. As time progresses, these mines advance outward to the other energy rings, ultimately to be released into space, whereupon they employ a primitive artificial intelligence to chase after the spaceship.

Mines cannot be destroyed by other mines or the cannon’s plasma balls, but they are destroyed by a direct hit from the spaceship’s missiles. When this occurs, the affected mine explodes and two new mines are generated on the innermost energy ring (up to a maximum of twelve active mines on screen at any one time). All mines are also destroyed when the cannon is destroyed, with only three mines appearing with the replacement cannon. All active mines remain active after the spaceship is destroyed, plaguing the next ship as soon as it appears.

Mines simulate the actions of heat-seeking missiles, always moving towards the spaceship, but at a pace that is somewhat slower than the ship. In addition, the mines exhibit a flocking behavior that encourages them to maintain a minimal distance from each other. As they gain proximity to their quarry (the spaceship), this separation tendency becomes less dominant.

Once released from the outermost energy ring, active mines may never reenter the star castle. As they chase the spaceship, their paths are never allowed to cross into the interior of the energy rings.

Like the cannon’s plasma balls, the mines cannot wrap around the screen, exiting from one side and emerging from the opposite side. Only the spaceship and its missiles are able to do this, providing the user with a strategic advantage that can be exploited to avoid destruction and to plan a course of attack.

The user may use the “P” key to pause the game’s animation. Although not implemented in the provided executable, an additional desired feature of the game is the ability to play in full-screen or window mode.
Your assignment is to demonstrate how design patterns may be applied to this game program. Specifically, there are five aspects of this program that commonly cause problems and that would benefit from the application of specific design patterns:

1. The various objects in the game (the spaceship, the cannon, the energy ring bars, the missiles, the mines, and the plasma balls) are commonly implemented with data members that flag their current condition (e.g., an energy ring bar has been hit zero, one, or two times; the spaceship may have collided with a mine, a missile, or an energy ring bar; the cannon’s aim is aligned with the spaceship and three destroyed energy ring bars, with the spaceship and less than three destroyed energy ring bars, or not with the spaceship at all). This type of implementation usually results in elaborate conditional statements that are distributed throughout the code, increasing the probability for errors.

2. The game logic and the presentation mode used (e.g., windowed mode vs. full-screen mode) are often combined into a single class, in spite of the fact that they are really quite independent. While a game might be intended for full-screen mode, developers often find debugging easier in windowed mode, but in either case, the logic of the game is unaffected by the presentation.

3. Animation is handled by updating the screen on a periodic basis. Keyboard events (like using the arrow keys to propel or spin the spaceship) are frequently used to directly update the position or orientation of the displayed object. Unfortunately, this often results in animation that is less than smooth, since the rate at which keyboard operations take place may not synchronize well with the automatic refresh rate of the screen.

4. Some of the game’s displayed objects (e.g., the spaceship, the cannon) lend themselves to being rendered by means of geometric primitives (i.e., line segments drawn in a particular pattern, but reoriented according to what the object is currently doing), while other objects (e.g., the plasma balls, the mines) are much more simplistic, lending themselves to a traditional “sprite” approach, in which a series of images are flipped through in sequence in order to implement their animation. If the sprite logic and the geometric primitive logic are combined into a single animation module, alterations would be rather difficult to make.

5. Collisions between objects are handled by means of bounding circles. For instance, if the smallest circle containing a particular missile overlaps the smallest circle containing the spaceship, then it is concluded that the missile and the spaceship have collided, resulting in the spaceship’s disintegration. However, with six different types of objects in the game (some of which, like the missiles, the mines, and the energy ring bars, have multiple instances), pairwise collision determination is rather difficult to implement.

(10 points each) For each of the five problems identified above, specify a design pattern that would alleviate the problem, using a paragraph or two to explain your selection and providing a detailed UML class diagram in each case to illustrate your explanation.

Pseudocode for a standard game loop:

while the game is running
  • update the game elements (the displayed objects)
  • check for and process any interactions (collisions)
  • render the game’s revised state to the back buffer
  • make the back buffer visible (swap buffers)

This assignment is due on your drop-box by 9 AM on Thursday, April 23, 2009.