Space Invaders - Iterative Enhancement Plan (IEP)

For Person 1

Throughout, help teammates as needed. Use methods and instance variables per the UML class diagram. Delete temporary code when continuing to the next stage. **Coordinate with teammates whenever you touch the shared files: *main*, *Controller*, *Game* and *View*. Commit-and-push at each stage but beware of Git conflicts!**

1. [**Entire team,** with your instructor] [*Most of the code for this stage is included in the starting code.*] A blank screen appears. A message appears on the Console that a Model (Game), View and Controller have been constructed. Events are (temporarily) printed on the Console. The background, title and size are set. Files contain team member names. [After this stage, the code is a bare-bones Model-View-Controller architecture.]

2. There is a ***Fighter*** class (in a *Fighter.py* file) that has stubs per the UML class diagram. The constructor method (*__init__*) sets instance variables per parameters and (temporarily) prints a simple message.

3. The Fighter appears, centered horizontally and near the bottom of the screen, drawn an image in the ***Assets*** folder. Sub-stages:
   a. The Game's ***draw_game*** method calls the Fighter's ***draw*** method, which just *prints* "draw fighter", temporarily.
   b. The Fighter appears on the screen at a hard-coded place.
   c. The Fighter shows up on the screen as specified.

4. The Fighter moves left/right 5 pixels per game loop cycle whenever the left/right arrow key is in the PRESSED state, but restricted so that the Fighter does not go more than ½ of the Fighter's width off the left or right edges of the screen. Sub-stages:
   a. Left/right keys just *print* "left", "right".
   b. Left/right keys make the Fighter move left/right.
   c. Fighter does not go off the screen, per the specification.

Coordinate the next stages with Person 3 (Missile class implementor).

5. When the Space bar is pressed, the Fighter fires a Missile, which causes a "pew" sound. (*Holding* the Space bar down does NOT generate additional Missile objects.) For now, no Missile appears; just the "pew" sound, which is via a sound file in the ***Assets*** folder.

Before continuing to the next stage, help Person 3 complete their work through their Stage 6, if they have not already completed that stage.

6. Each Missile appears on the screen centered at the Fighter horizontally, with the top of the Missile at the same y-position as the Fighter, in response to pressing the space bar. [To test multiple Missiles, move the Fighter while firing.]

7. To be continued per instructions given at the next session.

After completing the above, help teammates with their work.

Reminders:

1. ***Image / blit:***

```
self.image = pygame.image.load("../assets/fighter.png")
self.screen.blit(self.image, (self.x, self.y))
```

2. ***Draw line:***

```
pygame.draw.line(self.screen, self.color, (self.x, self.y),
    (self.x, self.y + self.height), self.width)
```

3. ***Play sound:***

```
self.fire_sound = pygame.mixer.Sound("../assets/pew.wav")
self.fire_sound.play()
```

4. ***Key interaction:***

```
if pressed_keys[pygame.K_LEFT]:  or  if key_was_pressed_...
```

Space Invaders - Iterative Enhancement Plan (IEP)

For Person 2

Throughout, help teammates as needed. Use methods and instance variables per the UML class diagram. Delete temporary code when continuing to the next stage. **Coordinate with teammates whenever you touch the shared files: *main*, *Controller*, *Game* and *View*. Commit-and-push at each stage but beware of Git conflicts!**

1. [**Entire team,** with your instructor] [*Most of the code for this stage is included in the starting code.*] A blank screen appears. A message appears on the Console that a Model (Game), View and Controller have been constructed. Events are (temporarily) printed on the Console. The background, title and size are set. Files contain team member names. [After this stage, the code is a bare-bones Model-View-Controller architecture.]

2. There is an ***Enemy*** class (in an *Enemy.py* file) that has stubs per the UML class diagram. The constructor method (*__init__*) sets instance variables per parameters and (temporarily) prints a simple message.

3. A single, temporary Enemy appears, somewhere near the top of the screen. Enemies are images in the ***Assets*** folder. The *Game* object constructs the temporary Enemy. Sub-stages:
   a. The Game's ***draw_game*** method calls the Enemy's ***draw*** method, which just *prints* "draw enemy", temporarily.
   b. The Enemy itself shows up on the screen as specified.

4. At each cycle of the game loop, the Enemy moves sideways (to the right) 5 pixels, and when it gets 100 pixels from where it starts, it moves down 10 pixels and reverses horizontal direction. Substages:
   a. Move sideways (to the right).
   b. Move down and reverse direction when 100 pixels from its start.

5. There is an ***Enemies*** class (in an *Enemies.py* file) that has stubs per the UML class diagram. The constructor method (*__init__*) sets instance variables per parameters and (temporarily) prints a simple message.

6. A two-dimensional array of Enemies appears, centered horizontally, near the top of the screen, in 5 rows of 8 Enemy objects per row. The Enemies object constructs and stores the Enemy objects. The temporary Enemy is removed.

7. Enemies move and are removed as needed. Sub-stages:
   a. Each Enemy moves just like the temporary Enemy moved.
   b. An Enemy is removed from the Enemies object's list of Enemy objects when the Enemy goes below the bottom of the screen. [Test this by printing the length of the Enemies list and being sure that it reduces as Enemies go below the bottom of the screen.]

8. To be continued per instructions given at the next session.

After completing the above, help teammates with their work.

Reminders:

5. Image / blit:

```
self.image = pygame.image.load("../assets/fighter.png")
self.screen.blit(self.image, (self.x, self.y))
```

6. Draw line:

```
pygame.draw.line(self.screen, self.color, (self.x, self.y),
    (self.x, self.y + self.height), self.width)
```

7. Play sound:

```
self.fire_sound = pygame.mixer.Sound("../assets/pew.wav")
self.fire_sound.play()
```

8. Key interaction:

```
if pressed_keys[pygame.K_LEFT]:  or  if key_was_pressed_...
```

Space Invaders - Iterative Enhancement Plan (IEP)

For Person 3

Throughout, help teammates as needed. Use methods and instance variables per the UML class diagram. Delete temporary code when continuing to the next stage. **Coordinate with teammates whenever you touch the shared files: *main*, *Controller*, *Game* and *View*. Commit-and-push at each stage but beware of Git conflicts!**

1. [**Entire team,** with your instructor] [*Most of the code for this stage is included in the starting code.*] A blank screen appears. A message appears on the Console that a Model (Game), View and Controller have been constructed. Events are (temporarily) printed on the Console. The background, title and size are set. Files contain team member names. [After this stage, the code is a bare-bones Model-View-Controller architecture.]

2. There is a ***Missile*** class (in a *Missile.py* file) that has stubs per the UML class diagram. The constructor method (*__init__*) sets instance variables per parameters and (temporarily) prints a simple message.

3. A single, temporary Missile appears, somewhere near the bottom of the screen. Missiles are drawn as filled red lines, 4 pixels wide and 8 pixels tall. The *Game* object constructs the temporary Missile (temporarily). Sub-stages:

   a. The Game's ***draw_game*** method calls the Missile's ***draw*** method, which just *prints* "draw missile", temporarily.

   b. The Missile itself shows up on the screen as specified.

4. At each cycle of the game loop, the Missile moves up 5 pixels.

5. Two Missiles appear, at different places on the screen. The temporary Missile is removed. [Implement this stage in the ***Missiles*** class, with the Missiles object constructing the two new temporary Missile objects. Implement and use the Missiles object's *add_missile* method to add a Missile to the Missiles object's list of Missiles.]

Sometime after completing the above stage, Person 1 will coordinate with you to implement their Stage 6.

6. Missiles move. Sub-stages:

   a. Each Missile in the Missiles list moves just like the temporary Missile moved.

   b. A Missile is removed from the Missiles object's list of Missile objects when the Missile goes above the top of the screen. [Test this by printing the length of the Missiles list and being sure that it reduces as Missiles go above the top of the screen.]

9. To be continued per instructions given at the next session.

After completing the above, help teammates with their work.

Reminders:

9. Image / blit:

```
self.image = pygame.image.load("../assets/fighter.png")
self.screen.blit(self.image, (self.x, self.y))
```

10. Draw line:

```
pygame.draw.line(self.screen, self.color, (self.x, self.y),
    (self.x, self.y + self.height), self.width)
```

11. Play sound:

```
self.fire_sound = pygame.mixer.Sound("../assets/pew.wav")
self.fire_sound.play()
```

12. Key interaction:

```
if pressed_keys[pygame.K_LEFT]:  or  if key_was_pressed_...
```