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!**

0. [**Starting code**] A blank screen appears. The code is a bare-bones Model-View-Controller architecture.

1. [**Entire** team**,** with your instructor] A blank screen appears. The background, caption and screen size are set. There are files with stubs for all classes (no methods or instance variables yet) and an instance of each object is constructed in Game's *__init__* method.

2. The ***Fighter*** class (in the *Fighter.py* file) has stubs for all its methods, per the UML class diagram, with each method having only its *self* parameter at this point. The constructor method (*__init__*) sets instance variables, per the UML class diagram, with temporary values (e.g. *None*) so far, and (temporarily) prints a simple message.

3. The Fighter appears, centered horizontally and near the bottom of the screen, drawn from 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 appears 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 more than ½ of its width off the screen.

Help Person 3 (Missile class implementor) complete their Stage 5 *while doing your own Stage 5*:

5. When the Space bar is pressed, the Fighter fires a Missile, which:
   a. Causes a "pew" sound (from the ***assets*** folder),
   b. Constructs a new Missile at the current position of the top of the Fighter, centered horizontally on the Fighter, and
   c. Adds that Missile to Missiles class (per the *add_missile* method in the Missiles class). This step also requires that the *__init__* for the Fighter have the Missiles object as a parameter.

   Pressing the space bar repeatedly causes multiple Missile objects to appear at the Fighter (and move). *Holding* the Space bar down does NOT generate additional Missile objects.

After completing Stage 5, help Persons 2 and 3 complete their Stage 5 and preceding (if they have not already completed them).

Then:

6. The Fighter plays an explosion sound when it is hit by an Enemy and the game ends. Sub-stages:
   a. Implement the "Fighter's *is_hit_by* and *explodes* methods, per their specifications.
   b. The Game object checks whether the Fighter is hit by an Enemy and reacts by (put the rest in a helper method) exploding the Fighter, printing a "You lose" message on the Console, playing a "lose" sound, and pausing the game at this stage (i.e., it no longer does the *run_one_cycle* actions).

After completing the above, help teammates complete their work.

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!**

0. [**Starting code**] A blank screen appears. The code is a bare-bones Model-View-Controller architecture.

1. [**Entire** team**,** with your instructor] A blank screen appears. The background, caption and screen size are set. There are files with stubs for all classes (no methods yet) and an instance of each object is constructed in Game's *__init__* method.

2. The ***Enemy*** class (in the *Enemy.py* file) has stubs for all its methods, per the UML class diagram, with each method having only its *self* parameter at this point. The constructor method (*__init__*) sets instance variables, per the UML class diagram, with temporary values (e.g. *None*) so far, and (temporarily) prints a simple message.

3. A single temporary Enemy appears on the screen as specified, drawn from an image in the ***Assets*** folder. Sub-stages:
   a. The Game's ***draw_game*** method calls the Enemy's ***draw*** method, which just *prints* "draw enemy", temporarily.
   b. The Enemy appears on the screen somewhere near the top of the screen, with *__init__* having optional parameters for its *x* and *y*.

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

5. Enemies are drawn and move, per the *Enemies* class. Sub-stages:
   a. The **Enemies** class (note plural) has stubs for its methods, per the UM class diagram.
   b. The constructor method (*__init__*) sets instance variables per parameters and creates, draws and moves a single temporary Enemy.
   c. 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 list of Enemy objects. The temporary Enemy is removed.
   d. Enemies move just like the temporary Enemy moved.

Do Stage 6 in coordination with Person 3:

6. When an Enemy is hit by a Missile, the Enemy explodes (that is, it plays an explosion sound and is removed from the Enemies list of Enemy objects, hence no longer is drawn/moves). Sub-stages:
   a. Implement the *is_hit_by* and *explode* methods in the Enemy class, per their specifications.
   b. Implement the *remove_exploded_enemies* in the Enemies class, per its specification.

7. If all Enemy objects are exploded, the game ends. That is, the Game object checks whether all the Enemies have exploded and reacts by (put the rest in a helper method) printing a "You win" message on the Console, playing a "win" sound, and pausing the game at this stage (i.e., it no longer does the *run_one_cycle* actions).

8. Enemy objects are 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.

After completing the above, help teammates complete their work.

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!**

0. [**Starting code**] A blank screen appears. The code is a bare-bones Model-View-Controller architecture.

1. [**Entire** team**,** with your instructor] A blank screen appears. The background, caption and screen size are set. There are files with stubs for all classes (no methods yet) and an instance of each object is constructed in Game's *__init__* method.

2. The ***Missile*** class (in the *Missile.py* file) has stubs for all its methods, per the UML class diagram, with each method having only its *self* parameter at this point. The constructor method (*__init__*) sets instance variables, per the UML class diagram, with temporary values (e.g. *None*) so far, and (temporarily) prints a simple message.

3. A single temporary Missile appears on the screen as specified, drawn from an image in the ***Assets*** folder. Sub-stages:
   a. The Game's ***draw_game*** method calls the Missile's ***draw*** method, which just *prints* "draw missile", temporarily.
   b. The Missile appears on the screen somewhere near the bottom of the screen, drawn as a *Rectangle*, with *__init__* having optional parameters for its *x*, *y*, *color*, *width* and *height*.

4. At each cycle of the game loop, the Missile moves up per its speed.

Help Person 1 (Fighter class implementor) complete their Stage 5 *while doing your own Stage 5*:

5. Missiles are drawn and move, per the ***Missiles*** class (note plural). Sub-stages:
   a. The Missiles class has stubs for its methods, per the UM class diagram.
   b. Two temporary Missile objects are in the *missiles_list*, at different places on the screen. Each Missile in the *missiles_list* is drawn and moves just like the temporary Missile moved.
   c. The *add_missile* method is implemented. The temporary Missile objects are removed. [Now test using the Fighter's *fire* method.]

After completing Stage 5, help Persons 1 and 2 complete their Stage 5 and preceding (if they have not already completed them).

Then do Stage 6 in coordination with Person 2:

6. Missiles explode and cause Enemies to explode when they hit an Enemy. Sub-stages:
   a. Implement the *is_hit_by* and *explode* methods in the Enemy class, per their specifications.
   b. Implement the *remove_exploded_enemies* in the Enemies class, per its specification.
7. 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.]

After completing the above, help teammates complete their work.