Phase IV

Reilly, Terrence Justice, James Goodall, Brad
Maresh, Keefer Gleason, Meagon

March 26, 2017

Contents

Implementation Languages

Our implementation was completed entirely in Python, using the library *pygame*.

Classes/Functions Implemented

- base_classes.py
  - RenderedBase 48 lines
  - ActorBase 24 lines
- ships.py
  - Player 5 lines
  - Enemy 4 lines
  - LaserBlaster 6 lines
  - LaserBeam 4 lines
- menu.py
  - Menu 92 lines
- driver.py
  - main 21 lines
- assets.py
  - GameSound 31 lines

Classes/Functions Not Implemented

No classes were unimplemented for this phase. All 12 Unit Tests are currently passing. However, there are still no functional or integration tests. So, some work remains to be done with integration.

References

Phase 4 required no external references for our team.

Appendix

A. Tested, and documented (internal comments) program listings

base_classes.py

Base classes listed below implement the functionality common to actionable objects and rendered objects. Most of our unit tests are concerned with these.

```python
import pygame


class RenderedBase(object):
    """Represents an object which can be rendered to the screen.

    Movements should be implemented in the subclass.
    """

    # The images should map a given action (as a String)
    # to a list of images.
    images = dict()

    # Should map a given action/event as a String
    # to a single sound.
    sounds = dict()

    # The current action being performed.
    _action = None
    _action_i = -1

    # The bounding box for this figure
    box = pygame.Rect(0, 0, 0, 0)

    def set_action(self, action):
        """Set the current action for this renderable object."""
        if action not in self.images:
            raise Exception('Action not defined for {}'.format(
                self.__name__
                ))
        self._action_i = 0
        self._action = action

    def __next__(self):
        """Get the next image to render to the screen.

        The action must first be set, and then next can be called upon it:
            some_actor = Actor()
            some_actor.set_action('walk')
            display(next(some_actor))

```

```
        Where Actor inherets RenderedBase, and display shows the returned
        image.
        """
        if self._action_i == -1:
            raise Exception('Action must be set')
        self._action_i += 1
        if self._action_i == len(self.images[self._action]):
            self._action_i = 0
        return self.images[self._action][self._action_i - 1]

    def render(self, context):
        """This should probably be updated once we have images."""
        pygame.draw.rect(context, (255, 0, 0), self.box)


class ActorBase(object):
    """Implements basic attributes of an actor."""

    def __init__(self, health=0, weapons=list()):
        self.health = health
        self.weapons = weapons
        self._weapon_i = -1 if len(self.weapons) == 0 else 0

    def add_weapon(self, weapon):
        """Adds a weapon to this actor's arsenal."""
        if self._weapon_i == -1:
            self._weapon_i = 0
        self.weapons.append(weapon)

    @property
    def weapon(self):
        """Get the current weapon for this player."""
        if not (0 <= self._weapon_i < len(self.weapons)):
            raise Exception('No weapons')
        return self.weapons[self._weapon_i]

    def next_weapon(self):
        """Switch to the next weapon."""
        self._weapon_i = (self._weapon_i + 1) % len(self.weapons)
```

driver.py

The driver file is a convenience for starting the program. It runs the main menu.

```
# This module initializes pygame and runs the main loop for the
# game.

import pygame
from lazer_blast import settings
```

```python
from lazer_blast.scenes import Menu


if __name__ == '__main__':
    pygame.init()
    windowSurface = pygame.display.set_mode(
        settings.SCREEN_DIMENSIONS, 0, 32
        )
    pygame.display.set_caption('Lazer Blast!')
    menu = Menu(windowSurface)
    menu.run()
```

scenes.py

The scenes file contains the main menu and the game itself, our two main scenes.

```python
# This module contains scenes that can be updates interchangeably
# within the main loop of the game.
import pygame
from lazer_blast import settings
from lazer_blast.ships import Player


class Game(object):
    """ The scene containing gameplay. """

    def __init__(self, screen):
        self.enemies = []
        self.beams = []
        self.player = Player()
        self.background = None
        self.running = True
        self.screen = screen
        self.clock = pygame.time.Clock()

    def handle_keydown(self, key):
        if key == pygame.K_a:
            self.player.momentum = (-1, self.player.momentum[1])
        elif key == pygame.K_s:
            self.player.momentum = (self.player.momentum[0], 1)
        elif key == pygame.K_d:
            self.player.momentum = (1, self.player.momentum[1])
        elif key == pygame.K_w:
            self.player.momentum = (self.player.momentum[0], -1)
        elif key == pygame.K_SPACE:
            print('space pressed')
        elif key == pygame.K_e:
            print('e pressed')
        elif key == pygame.K_q:
```

```
            print('q pressed')
        elif key == pygame.K_ESCAPE:
            self.running = False

    def handle_keyup(self, key):
        if key == pygame.K_a:
            self.player.momentum = (0, self.player.momentum[1])
        elif key == pygame.K_s:
            self.player.momentum = (self.player.momentum[0], 0)
        elif key == pygame.K_d:
            self.player.momentum = (0, self.player.momentum[1])
        elif key == pygame.K_w:
            self.player.momentum = (self.player.momentum[0], 0)

    def run(self):
        self.running = True
        settings.ITEMS = ('Resume', settings.ITEMS[1], settings.ITEMS[2])
        while self.running:
            # Limit frame speed to 60 FPS
            self.clock.tick(settings.FPS)

            # Handle key presses
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    self.running = False
                if event.type == pygame.KEYDOWN:
                    self.handle_keydown(event.key)
                if event.type == pygame.KEYUP:
                    self.handle_keyup(event.key)

            # Render items
            self.screen.fill(settings.BG_COLOR)

            for enemy in self.enemies:
                enemy.render(self.screen)
            self.player.render(self.screen)

            # Once there are images, this is what should be rendered
            next(self.player)

            pygame.display.flip()


class MenuActions:
    """Actions.  Should match the order of items in settings.ITEMS."""
    GAME = 0
    HIGH_SCORES = 1
    EXIT = 2
```

```python


class _MenuItems(object):

    def __init__(self):
        # Represents the current item selected (which should match
        # MenuActions.)
        self.current = 0
        self.font = pygame.font.SysFont(
            settings.FONT,
            settings.FONT_SIZE,
            )

    def next(self):
        self.current += 1
        if self.current == len(settings.ITEMS):
            self.current = 0

    def prev(self):
        self.current -= 1
        if self.current < 0:
            self.current = len(settings.ITEMS) - 1

    def render(self):
        ret = list()
        for index, item in enumerate(settings.ITEMS):
            curr = ''
            if index == self.current:
                curr = '>    {}'.format(item)
            else:
                curr = '      {}'.format(item)

            label = self.font.render(curr, 1, settings.FONT_COLOR)

            width = label.get_rect().width
            height = label.get_rect().height

            posx = (settings.SCREEN_WIDTH / 2) - (width / 2)
            totalHeight = len(settings.ITEMS) * height
            posy = (
                (settings.SCREEN_HEIGHT / 2)
                - (totalHeight / 2)
                + (index * height)
                )

            ret.append([label, (posx, posy)])
        return ret

```

```python

# The menu class implements the main menu for the game, which can
# in turn navigate to the game, a future high score table once the
# scoring system is in place within the game, as well as quitting
# the application.

class Menu(object):
    """The scene containing the title screen and options."""

    def __init__(self, screen):
        self.screen = screen
        self.clock = pygame.time.Clock()
        self.menu_items = _MenuItems()
        self.running = True
        self.game = Game(screen)

    def item_select(self, key):
        if key == pygame.K_UP:
            self.menu_items.prev()
        elif key == pygame.K_DOWN:
            self.menu_items.next()
        elif key == pygame.K_SPACE:
            if self.menu_items.current == MenuActions.GAME:
                self.game.run()
            elif self.menu_items.current == MenuActions.HIGH_SCORES:
                pass
            if self.menu_items.current == MenuActions.EXIT:
                self.running = False

    def run(self):
        """Run the menu.

        If the game is exited, the menu will continue running.
        If the menu is exited, then the game ends.
        """
        while self.running:
            # Limit frame speed to 60 FPS
            self.clock.tick(settings.FPS)

            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    self.running = False
                if event.type == pygame.KEYDOWN:
                    self.item_select(event.key)

            # Redraw the background
            self.screen.fill(settings.BG_COLOR)

```

```
            for label, position in self.menu_items.render():
                self.screen.blit(label, position)

            pygame.display.flip()
```

settings.py

The settings file contains global settings we might like to change easily (or that users would like to change.)

```
FPS = 60
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600
SCREEN_DIMENSIONS = (SCREEN_WIDTH, SCREEN_HEIGHT)

# Menu Settings
BG_COLOR = (0, 0, 0)
FONT = 'Arial'
FONT_SIZE = 30
FONT_COLOR = (255, 255, 255)
ITEMS = ('Start Game', 'High Scores', 'Quit')

# Player Settings
SPEED = 5
```

ships.py

The ships file contains the player and enemies.

```
import pygame

from lazer_blast import settings
from lazer_blast.base_classes import RenderedBase, ActorBase


class Player(ActorBase, RenderedBase):
    """ Player's ship """

    # TODO: Brad's images go here
    images = {
        'box': [None, None],
        }

    def __init__(self, controls=dict(), health=0, weapons=list()):
        self.health = health
        self.weapons = weapons
        self._weapon_i = -1 if len(self.weapons) == 0 else 0
        self.controls = controls
        self.position = (0, 0)
        self.box = pygame.Rect(0, 0, 50, 50)
```

```
        self.set_action('box')

        # Describes how fast the player is moving in a given direction.
        self.momentum = (0, 0)

    def _update_position(self):
        """Update the Player's position."""
        self.box = self.box.move(*(settings.SPEED * x for x in self.momentum))

    def __next__(self):
        """Update the position, and return the next image in the
        sequence for this action."""
        self._update_position()
        return super(Player, self).__next__()


class Enemy(ActorBase, RenderedBase):
    """ Enemy ship that combats player. """

    def __init__(self, health=0, weapons=list()):
        ActorBase.__init__(health=health, weapons=weapons)


class LaserBlaster(object):
    """ A weapon for ships to use. """

    def __init__(self, color=(0, 0, 0), charge=0, damage=0):
        self.color = color
        self.charge = charge
        self.damage = damage


class LaserBeam(RenderedBase):
    """ Graphic representation of a fired laserbeam. """

    def __init__(self, color=(0, 0, 0)):
        self.color = color
```

test_abstracts.py

The *test_abstracts* file tests the assumptions underlying the base classes. In particular, it ensures that the magic methods function.

```
"""Tests for the abstract base classes for our game."""
import unittest
from unittest import mock
import pygame

from lazer_blast.base_classes import (
```

```
    ActorBase,
    RenderedBase,
    )


class RenderedBaseTestCase(unittest.TestCase):
    # A fake subclass
    class Pantomime(RenderedBase):
        images = {
            'walk': ['fake_walk_{}'.format(x) for x in range(10)],
            'run': ['fake_run_{}'.format(x) for x in range(10)],
            }
        sounds = {
            'bump': 'fake_bump'
            }

    def test_has_images(self):
        rb = RenderedBase()
        self.assertTrue(isinstance(rb.images, dict))

    def test_has_sound(self):
        rb = RenderedBase()
        self.assertTrue(isinstance(rb.sounds, dict))

    def test_subclass_can_access_sequence_of_images(self):
        pant = self.Pantomime()

        pant.set_action('walk')
        self.assertEqual(next(pant), 'fake_walk_0')
        self.assertEqual(next(pant), 'fake_walk_1')
        self.assertEqual(next(pant), 'fake_walk_2')

        pant.set_action('run')
        self.assertEqual(next(pant), 'fake_run_0')

    def test_not_setting_action_raises_exception(self):
        pant = self.Pantomime()
        with self.assertRaises(Exception):
            next(pant)

    def test_undefined_action_raises_exception(self):
        pant = self.Pantomime()
        with self.assertRaises(Exception):
            pant.set_action('skeddadle')

    def test_bounding_box_defined(self):
        rb = RenderedBase()
        self.assertTrue(isinstance(rb.box, pygame.Rect))
```

```python

    @mock.patch('lazer_blast.base_classes.pygame.draw.rect')
    def test_render_called_with_box(self, mock_draw):
        surface = pygame.Surface((1000, 1000))
        rb = RenderedBase()
        rb.render(surface)
        self.assertEqual(mock_draw.call_count, 1)
        positional, _ = mock_draw.call_args
        self.assertEqual(
            positional[2],
            rb.box,
            )


class ActorBaseTestCase(unittest.TestCase):

    def test_actor_base_has_health(self):
        actor = ActorBase()
        self.assertTrue(isinstance(actor.health, int))

    def test_actor_can_have_weapons(self):
        actor = ActorBase()
        self.assertTrue(isinstance(actor.weapons, list))

    def test_can_add_weapons_to_actor(self):
        weapon1, weapon2 = 'A fake weapon', 'Another fake weapon'
        actor = ActorBase(100, [weapon1])
        self.assertEqual(actor.weapon, weapon1)
        actor.add_weapon(weapon2)
        actor.next_weapon()
        self.assertEqual(actor.weapon, weapon2)
        actor.next_weapon()
        self.assertEqual(actor.weapon, weapon1)

    def test_single_weapon_never_changes(self):
        weapon1 = 'A fake weapon'
        actor = ActorBase(100, [weapon1])
        actor.add_weapon(weapon1)
        self.assertEqual(actor.weapon, weapon1)
        actor.next_weapon()
        self.assertEqual(actor.weapon, weapon1)
```

test_ships.py

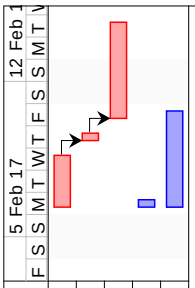
The *test_ships* file tests the player ship currently, and has some stubbed tests for enemies. It currently tests player movement through the *momentum* abstraction.

```python
"""Tests for the ships for our game."""
import unittest
```

```python
import pygame
import random

from lazer_blast import settings
from lazer_blast.ships import (
    Player,
    )


class PlayerTestCase(unittest.TestCase):

    def setUp(self):
        self.rand = random.Random()

    def test_set_momentum_moves_ship_on_update(self):
        player = Player()
        player.box = pygame.Rect(0, 0, 0, 0)
        player.momentum = (1, 0)
        ticks = self.rand.randint(3, 100)
        for i in range(ticks):
            next(player)
        expected = pygame.Rect(
            ticks * settings.SPEED,
            0, 0, 0
            )
        self.assertEqual(player.box, expected)


class EnemyTestCase(unittest.TestCase):

    def test_enemy_moves(self):
        self.assertTrue(False, 'Finish the test!')

    def test_enemy_stays_within_bounds(self):
        self.assertTrue(False, 'Finish the test!')
```

B. Gantt Chart for Phase 4

		Name	Duration	Start	Finish	Predecessors	Resource Names
1		Player Movements	3 days?	2/6/17 8:00 AM	2/8/17 5:00 PM		
2		Enemy Movements	1 day?	2/9/17 8:00 AM	2/9/17 5:00 PM	1	
3		Attacks	3 days?	2/10/17 8:00 AM	2/14/17 5:00 PM	2	
4		Sounds	1 day?	2/6/17 8:00 AM	2/6/17 5:00 PM		
5		Images	5 days?	2/6/17 8:00 AM	2/10/17 5:00 PM		



LazerBlast - page1