

CPS410 Project

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

April 19, 2017

CPS410
CS Department CMU

Contents

Phase I: Team Composition and Problem Definition	3
Phase II: SRS	4
Phase III: SD	5
Phase IV: Coding and Unit Testing	6
Phase V: System Integration and Testing	7
Phase VI: Demo	8
Summary and Conclusion	9
Bibliography	10
Appendices	11
Updated/final Functional, Object and Dynamic models	12
Gantt chart for the entire project one Gantt chart	13
Updated/final program listings	15
Data files used if any	40
Sample runs	41
Users manual	42
Articles Collected and Used if any	43

Phase I: Team Composition and Problem Definition

Phase II: SRS

Phase III: SD

Phase IV: Coding and Unit Testing

Phase V: System Integration and Testing

Phase VI: Demo











Summary and Conclusion

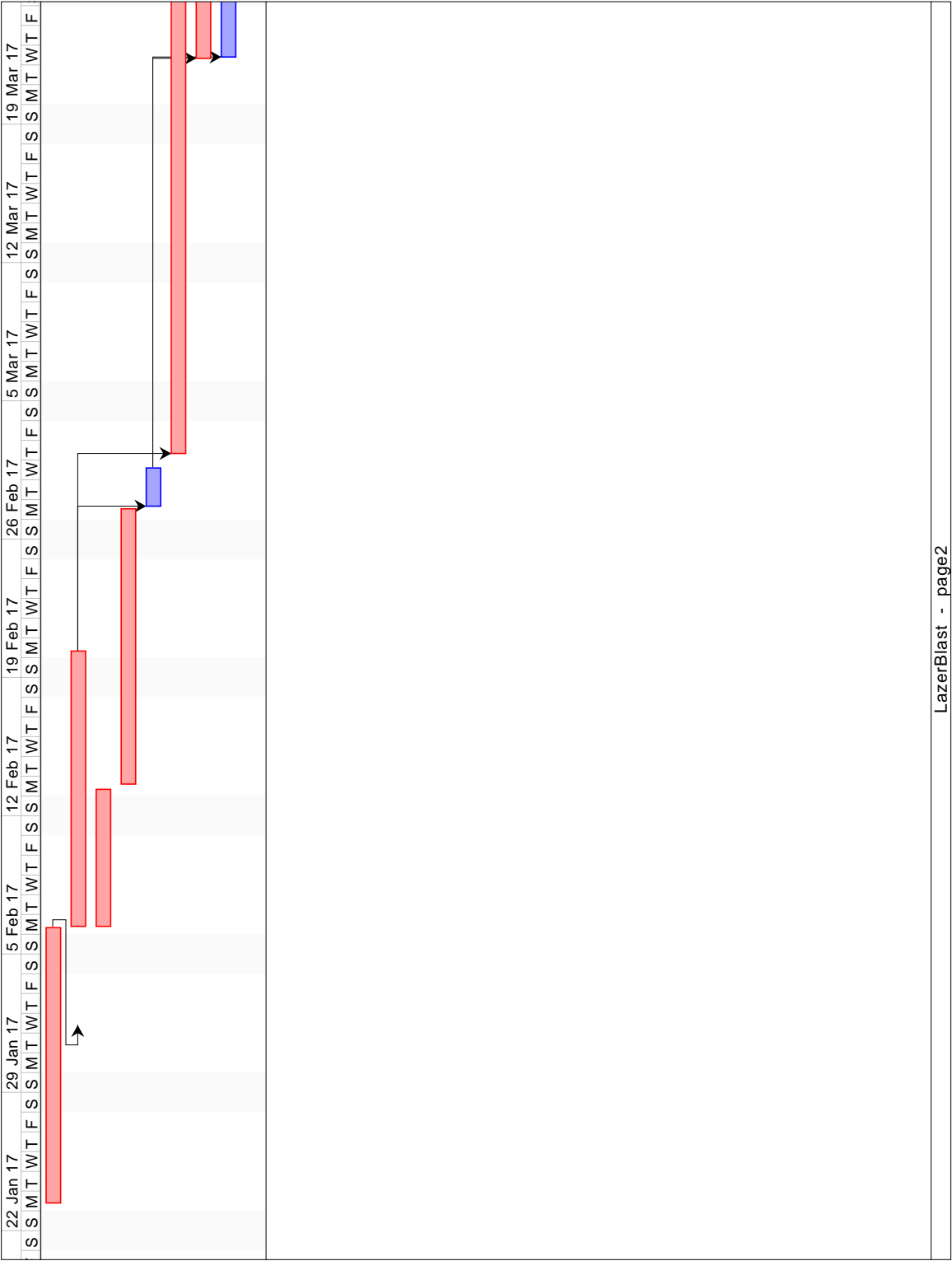
Bibliography

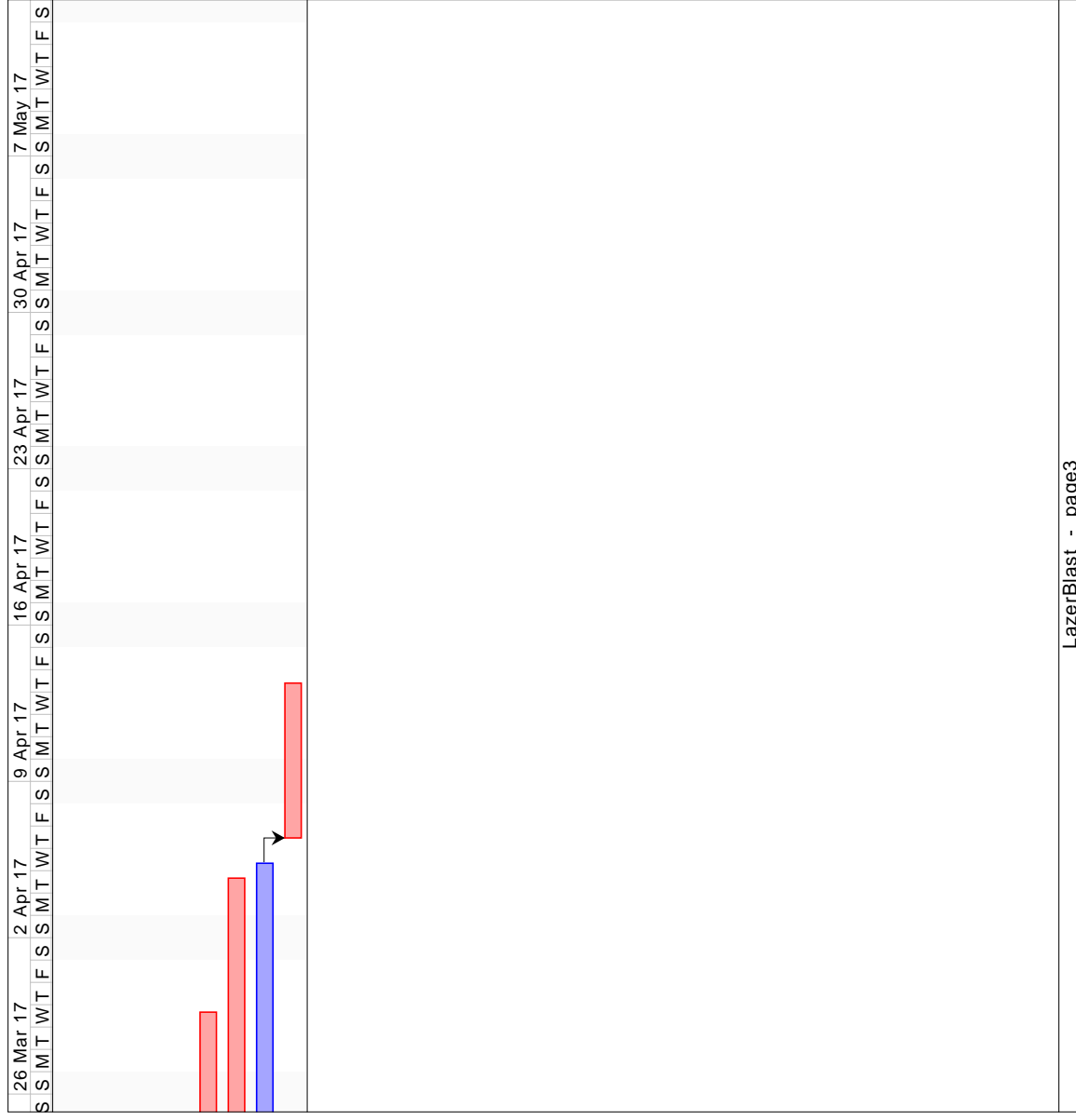
Appendices

Updated/final Functional, Object and Dynamic models

Gantt chart for the entire project one Gantt chart

		Name	Duration	Start	Finish	Predecessors	Resource Names	F
1		Base Classes	10 days	1/23/17 9:00 AM	2/6/17 9:00 AM			
2		Ships	10 days?	2/1/17 9:00 AM	2/15/17 9:00 AM	1		
3		Sound	5 days?	2/6/17 10:00 AM	2/13/17 10:00 AM			
4		Menu	10 days?	2/13/17 2:00 PM	2/27/17 2:00 PM			
5		Images	2 days?	2/27/17 4:00 PM	3/1/17 4:00 PM	2		
6		Unit Testing	20 days	3/2/17 8:00 AM	3/29/17 5:00 PM	2		
7		Functional Testing	10 days?	3/22/17 8:00 AM	4/4/17 5:00 PM	5		
8		Usability Testing	10 days?	3/22/17 9:00 AM	4/5/17 9:00 AM	5		
9		Deployment	5 days?	4/6/17 11:00 AM	4/13/17 11:00 AM	8		





Updated/final program listings

```
#--base_classes.py-----
from itertools import cycle
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()

    # A cycle of actions to be performed.
    _action = None
    # The current action being performed.
    _curr = None

    # The bounding box for this figure
    box = pygame.Rect(0, 0, 0, 0)
    surface = pygame.Surface((0, 0))
    color = (255, 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 = cycle(self.images[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 is None:
            raise Exception('Action must be set')
```

```

        self._curr = next(self._action)
        return self._curr

    def render(self, context):
        """This should probably be updated once we have images."""
        if self._curr is not None:
            context.blit(self._curr, self.box)
        else:
            pygame.draw.rect(context, self.color, self.box)

    def move(self, x, y):
        self.box = self.box.move(x, y)

    def in_bounds(self):
        """Return True if this actor is within the bounds of the surface"""
        return self.surface.get_bounding_rect().contains(self.box) == 1

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
        # Also holds bounds information
        self._position = pygame.Rect((0, 0, 100, 100))

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

```

```

import pygame
from lazer_blast import settings
from lazer_blast.scenes import Menu

def main():
    pygame.init()
    windowSurface = pygame.display.set_mode(
        settings.SCREEN_DIMENSIONS, 0, 32
    )
    pygame.display.set_caption('Lazer Blast!')
    menu = Menu(windowSurface)
    menu.run()

if __name__ == '__main__':
    main()
# This module contains scenes that can be updates interchangeably
# within the main loop of the game.
import os
import pygame
from random import Random
from time import time
import yaml

from lazer_blast import settings
from lazer_blast.ships import (
    Enemy,
    Player,
    HealthBar,
    ScoreBoard,
)

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

    def __init__(self, screen):
        self.surface = pygame.Surface(settings.SCREEN_DIMENSIONS)
        self.beams = []
        self.enemies = list()
        self.player = Player()
        self.player.surface = self.surface
        self.background = None
        self.running = True
        self.game_over = False
        self.screen = screen

```

```

self.clock = pygame.time.Clock()
self.rand = Random()
self.health_bar = HealthBar(player=self.player)
self.score_board = ScoreBoard()
self.spawn = settings.get_spawn_function(time())
next(self.player)

def generate_enemies(self):
    modified_spawn_rate = self.spawn(time())
    if self.rand.random() < modified_spawn_rate:
        x = self.rand.randint(0, settings.SCREEN_DIMENSIONS[0])
        color = self.rand.choice(settings.COLORS)
        enemy = Enemy(starting_pos=(x, 0), color=color)
        enemy.target = self.player
        enemy.surface = self.surface
        self.enemies.append(
            enemy
        )
        next(enemy)

def handle_keydown(self, key):
    if key == settings.LEFT:
        self.player.momentum = (-1, self.player.momentum[1])
    elif key == settings.DOWN:
        self.player.momentum = (self.player.momentum[0], 1)
    elif key == settings.RIGHT:
        self.player.momentum = (1, self.player.momentum[1])
    elif key == settings.UP:
        self.player.momentum = (self.player.momentum[0], -1)
    elif key == settings.FIRE:
        self.player.flip_laser(True)
    elif key == settings.SWAP_RIGHT:
        self.player.next_color()
    elif key == settings.SWAP_LEFT:
        self.player.prev_color()
    elif key == settings.ESCAPE:
        self.running = False

def handle_keyup(self, key):
    if key == settings.LEFT:
        self.player.momentum = (0, self.player.momentum[1])
    elif key == settings.DOWN:
        self.player.momentum = (self.player.momentum[0], 0)
    elif key == settings.RIGHT:
        self.player.momentum = (0, self.player.momentum[1])
    elif key == settings.UP:
        self.player.momentum = (self.player.momentum[0], 0)
    elif key == settings.FIRE:

```

```

        self.player.flip_laser(False)

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)
        self.handle_key_events()
        self.render_items()
        self.handle_collisions()
        self.score_board.add_points(
            sum([1 for x in self.enemies if x.health <= 0])
        )
        self.enemies = [x for x in self.enemies
                        if x.in_bounds() and x.health > 0]
        self.generate_enemies()

def handle_key_events(self):
    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)

def render_items(self):
    self.screen.fill(settings.BG_COLOR)
    for enemy in self.enemies:
        enemy.render(self.screen)
    self.player.render(self.screen)
    self.health_bar.render(self.screen)
    self.score_board.render(self.screen)

    # Once there are images, this is what should be rendered
    for enemy in self.enemies:
        next(enemy)
    next(self.player)
    next(self.health_bar)
    pygame.display.flip()

def handle_collisions(self):
    if self.player.enemies_touching(self.enemies):
        self.player.health -= settings.ENEMY_STRENGTH
        if self.player.health <= 0:
            self.player.PlaySound("275008__alienxxx__mayday-mayday.wav")
            pygame.time.wait(2700)

```

```

        self.player.PlaySound("35462__jobro__explosion-5.wav")
        self.running = False
        self.game_over = True

class HighScores(object):

    def __init__(self, screen):
        self.path = os.path.join(
            settings.BASE_DIR,
            'assets/high_scores.yaml',
        )
        self.screen = screen
        self.clock = pygame.time.Clock()
        self.scores = None
        self.running = True
        self.load()
        self.font = pygame.font.SysFont(
            settings.FONT,
            settings.FONT_SIZE,
        )

    def load(self):
        with open(self.path, 'r') as fin:
            self.scores = yaml.load(fin)
        if self.scores is None:
            self.scores = list()

    def add_score(self, score):
        self.scores.append(score * 10)
        self.scores.sort(reverse=True)
        self.scores = self.scores[:10]

    def save(self):
        with open(self.path, 'w') as fout:
            yaml.dump(self.scores, fout)

    def render(self):
        label = self.font.render('High Scores', 1, settings.FONT_COLOR)
        curr_y = settings.SCREEN_HEIGHT * 0.25
        x = settings.SCREEN_WIDTH * 0.5 - label.get_rect().width / 2
        self.screen.blit(label, (x, curr_y))
        curr_y += label.get_rect().height + 5
        for score in self.scores:
            label = self.font.render(str(score), 1, settings.FONT_COLOR)
            self.screen.blit(label, (x, curr_y))
            curr_y += label.get_rect().height + 5

```

```

def handle_key_events(self):
    # Any key
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            self.running = False

def run(self):
    self.running = True
    while self.running:
        self.clock.tick(settings.FPS)
        self.handle_key_events()
        self.screen.fill(settings.BG_COLOR)
        self.render()
        pygame.display.flip()

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

class _MenuItems(object):

    def __init__(self, game):
        # Represents the current item selected (which should match
        # MenuActions.)
        self.current = 0
        self.game = game
        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()
        items = None

```

```

if not self.game.game_over:
    items = settings.ITEMS
else:
    items = settings.GAME_OVER_ITEMS

for index, item in enumerate(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

```

*# 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.game = Game(screen)
        self.menu_items = _MenuItems(self.game)
        self.running = True
        self.high_scores = HighScores(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_RETURN:
    if self.menu_items.current == MenuActions.GAME:
        if not self.game.game_over:
            self.game.run()
        else:
            self.game = Game(self.screen)
            self.menu_items.game = self.game
            self.game.run()
        self.game.run()
        self.high_scores.add_score(self.game.score_board.score)
    elif self.menu_items.current == MenuActions.HIGH_SCORES:
        self.high_scores.running = True
        self.high_scores.run()
    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()
        self.high_scores.save()

#----settings.py-----
from math import sin
import os
import pygame

```

FPS = 60

```

SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600
SCREEN_DIMENSIONS = (SCREEN_WIDTH, SCREEN_HEIGHT)
COLORS = [
    pygame.color.THECOLORS[x]
    for x in ['red', 'green', 'blue']
]
COLOR_LOOKUP = {
    pygame.color.THECOLORS[x]: x
    for x in ['red', 'green', 'blue']
}

BASE_DIR = os.path.dirname(os.path.abspath(__file__))

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

# Player Settings
SPEED = 7
PLAYER_HEALTH = 100
PLAYER_STRENGTH = 1

# Enemy Settings
ENEMY_SPEED = 2
ENEMY_HEALTH = 5
SPAWN_RATE = 0.2
TURN_BOUND = 0.01
ENEMY_STRENGTH = 1

def get_spawn_function(start_time, compression=0.75):
    def spawn_function(x):
        return SPAWN_RATE + 0.2 * sin((x - start_time) * compression)
    return spawn_function

# Control Settings
LEFT = pygame.K_a
DOWN = pygame.K_s
RIGHT = pygame.K_d
UP = pygame.K_w
FIRE = pygame.K_SPACE
SWAP_RIGHT = pygame.K_RIGHT

```

```

SWAP_LEFT = pygame.K_LEFT
ESCAPE = pygame.K_ESCAPE

_dvorak = True # False
if _dvorak:
    LEFT = pygame.K_a
    DOWN = pygame.K_o
    RIGHT = pygame.K_e
    UP = pygame.K_COMMA
    FIRE = pygame.K_SPACE
    SWAP_RIGHT = pygame.K_PERIOD
    SWAP_LEFT = pygame.K_QUOTE
    ESCAPE = pygame.K_ESCAPE

#----ships.py-----
import os
from random import Random
import pygame

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

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

    images = {
        'fly': [
            pygame.image.load(os.path.join(settings.BASE_DIR, x))
            for x in ['assets/player.png']
        ],
    }

    def __init__(self, controls=dict(),
                 health=settings.PLAYER_HEALTH, weapons=list()):
        self.health = health
        self.weapons = weapons
        self._weapon_i = -1 if len(self.weapons) == 0 else 0
        self.controls = controls
        self.set_action('fly')
        self.box = self.images['fly'][0].get_rect()
        self.box.left = settings.SCREEN_WIDTH / 2 - 25
        self.box.top = settings.SCREEN_HEIGHT - 50
        self.color_i = 0
        self.color = settings.COLORS[self.color_i]
        self._laser = False
        self.laser = pygame.Rect((
            self.box.left + (0.5 * self.box.width) - 1,

```

```

        self.box.top - settings.SCREEN_HEIGHT,
        3,
        settings.SCREEN_HEIGHT,
    ))

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

def move(self, x, y):
    super(Player, self).move(x, y)
    if not self.in_bounds():
        height = self.surface.get_bounding_rect().height
        width = self.surface.get_bounding_rect().width
        diffx = 0
        if self.box.left < 0:
            diffx = - self.box.left
        elif self.box.right > width:
            diffx = width - self.box.right
        diffy = 0
        if self.box.top < 0:
            diffy = - self.box.top
        elif self.box.bottom > height:
            diffy = height - self.box.bottom
        self.box = self.box.move(diffx, diffy)
    self.laser.left = self.box.left + (0.5 * self.box.width) - 1
    self.laser.top = self.box.top - settings.SCREEN_HEIGHT + 4

def _update_position(self):
    """Update the Player's position."""
    self.move(*(x * settings.SPEED 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__()

def next_color(self):
    """Switch to the next color laser."""
    self.color_i += 1
    if self.color_i >= len(settings.COLORS):
        self.color_i = 0
    self.color = settings.COLORS[self.color_i]

def prev_color(self):
    """Switch to the previous color laser."""
    self.color_i -= 1
    if self.color_i < 0:

```

```

        self.color_i = len(settings.COLORS) - 1
        self.color = settings.COLORS[self.color_i]

def enemies_touching(self, enemies):
    """Return True if any of the given enemies is touching this player."""
    # We don't use collidelist() because we would have to gather
    # all of the rects first. This way we short-circuit.
    for enemy in enemies:
        if self.box.colliderect(enemy.box):
            Player.PlaySound(
                self, "367622_fxkid2_explosion-with-debris.wav")
            return True
    return False

def flip_laser(self, state=None):
    """Changes the state of the laser (on/off).
    Args:
    state: An optional state argument (True or False).
    If None, the laser will switch states.
    """
    self.PlaySound("42106_marcuslee_laser-wrath-4.wav")
    if state is None:
        self._laser = not self._laser
    else:
        # Make sure that _laser is a boolean value
        self._laser = bool(state)

def render(self, context):
    if self._laser:
        pygame.draw.rect(context, self.color, self.laser)
    return super(Player, self).render(context)

def PlaySound(self, file):
    assets_path = os.path.join(settings.BASE_DIR, "assets")
    sound_path = os.path.join(assets_path, file)
    pygame.mixer.init()
    pygame.mixer.music.load(sound_path)
    pygame.mixer.music.play()

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

    images = dict()

    possible_images = {
        x: pygame.image.load(
            os.path.join(

```

```

        settings.BASE_DIR,
        'assets/enemy_{}_1.png'.format(x)
    )
)
for x in ['red', 'green', 'blue']
}

def _set_images(self):
    """Since the color is dynamic at creation, the image is, too."""
    try:
        color = settings.COLOR_LOOKUP[self.color]
    except KeyError:
        color = 'red'
    self.images = {'fly': [self.possible_images[color]]}

def __init__(self, health=settings.ENEMY_HEALTH,
              color=settings.COLORS[0], starting_pos=(0, 0)):
    self.health = health
    self.weapons = list()
    self.color = color
    self.box = pygame.Rect(starting_pos[0], starting_pos[1], 50, 50)
    self._set_images()
    self.set_action('fly')
    self.momentum = (0, 1)
    self.rand = Random()
    # Target (player) must be set or enemy will not take damage
    self.target = None

def _update_position(self):
    """Update the Player's position."""
    dx = self.momentum[0]
    r = self.rand.random()
    if 0 <= r <= settings.TURN_BOUND:
        dx = -1
    elif 0.5 - settings.TURN_BOUND <= r <= 0.5 + settings.TURN_BOUND:
        dx = 0
    elif 1 - settings.TURN_BOUND <= r <= 1:
        dx = 1
    self.momentum = (dx, 1)
    self.move(*(x * settings.ENEMY_SPEED for x in self.momentum))

def _is_hit(self):
    if self.target is None:
        return False
    return (self.target._laser
            and bool(self.target.laser.collidirect(self.box))
            and self.color == self.target.color)

```

```

def _update_health(self):
    if self._is_hit():
        self.health -= settings.PLAYER_STRENGTH

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

def in_bounds(self):
    """Return True if this actor is within the bounds of the surface"""
    return self.surface.get_bounding_rect().collidepoint(
        self.box.topleft) == 1

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 laserbeamSRA. """

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

class HealthBar(RenderedBase):

    def __init__(self, color=(0, 255, 0), player=None):
        self.color = color
        self.box = pygame.Rect(
            settings.SCREEN_WIDTH * 0.25,
            settings.SCREEN_HEIGHT - 10,
            settings.SCREEN_WIDTH * 0.5,
            5,
        )
        self.player = player

    def __next__(self):
        """Updates for width to match the player's health."""
        if self.player is None:

```

```

        return
    percent = self.player.health / settings.PLAYER_HEALTH
    self.box.width = percent * settings.SCREEN_WIDTH * 0.5

class ScoreBoard(RenderedBase):

    def __init__(self, color=(255, 255, 255)):
        self.color = color
        self.font = pygame.font.SysFont(
            settings.FONT,
            settings.FONT_SIZE,
        )
        self.score = 0

    def add_points(self, amount=1):
        self.score += amount

    def _render_score(self):
        return 'Score: {}'.format(self.score * 10)

    def render(self, context):
        """This should probably be updated once we have images."""
        label = self.font.render(self._render_score(), 1, settings.FONT_COLOR)

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

        if width > 50:
            posx = settings.SCREEN_WIDTH - width - 10
        else:
            posx = settings.SCREEN_WIDTH - 50

        posy = (
            settings.SCREEN_HEIGHT
            - height
            - 5
        )
        context.blit(label, (posx, posy))

#----setup.py-----
from setuptools import setup, find_packages

with open('README.md', 'r') as fin:
    readme = fin.read()

setup(
    name='lazer_blast',

```



```

version='0.0.0',
description='A 2D game for our CPS410 class',
long_description=readme,
packages=find_packages(exclude=('tests', 'docs')),
entry_points={
    'console_scripts': [
        'lazerblast = lazer_blast.driver:main',
    ],
},
setup_requires=['pytest-runner'],
tests_require=['pytest']
)

#----abstract_tests.py-----
"""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))

    @unittest.skip('Why is this here? Sounds don\'t appear to be used.')
    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))

@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,
    )

def test_in_bounds(self):
    rb = RenderedBase()
    rb.box = pygame.Rect((0, 0, 100, 100))
    rb.surface = pygame.Surface((500, 500))
    self.assertTrue(rb.in_bounds())
    rb.box = pygame.Rect((-100, -100, -50, -50))
    self.assertFalse(rb.in_bounds())
    rb.box = pygame.Rect((-100, 100, -50, 50))
    self.assertFalse(rb.in_bounds())
    rb.box = pygame.Rect((500, 500, 600, 600))
    self.assertFalse(rb.in_bounds())

```

```

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)

```

```

#----ship_tests.py-----
"""Tests for the ships for our game."""
import unittest
from unittest import mock
import pygame
import random

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

```

```

class PlayerTestCase(unittest.TestCase):

    def setUp(self, surface=None):
        self.rand = random.Random()

    @mock.patch('lazer_blast.ships.pygame.mixer')

```

```

@mock.patch('lazer_blast.ships.os')
def test_set_momentum_moves_ship_on_update(self, mock_os, mock_mixer):
    player = Player()
    player.box = pygame.Rect(0, 0, 10, 10)
    player.surface = pygame.Surface((500, 500))
    player.momentum = (1, 0)
    ticks = self.rand.randint(3, 10)
    for i in range(ticks):
        next(player)
    expected = pygame.Rect(
        ticks * settings.SPEED,
        0, 10, 10
    )
    self.assertEqual(player.box, expected)

@mock.patch('lazer_blast.ships.pygame.mixer')
@mock.patch('lazer_blast.ships.os')
def test_cannot_move_player_out_of_bounds(self, mock_os, mock_mixer):
    player = Player()
    player.surface = pygame.Surface((500, 500))
    player.box = pygame.Rect((0, 0, 100, 100))
    self.assertEqual(player.box, pygame.Rect((0, 0, 100, 100)))
    player.move(10, 10)
    self.assertEqual(
        player.box, pygame.Rect((10, 10, 100, 100)),
        'This should be legal!'
    )
    player.move(-20, -10)
    self.assertNotEqual(
        player.box, pygame.Rect((-10, 0, 100, 100)),
        'This should have been illegal, and readjusts',
    )
    self.assertEqual(
        player.box, pygame.Rect((0, 0, 100, 100)),
        'It should readjust to be in position'
    )

@mock.patch('lazer_blast.ships.pygame.mixer')
@mock.patch('lazer_blast.ships.os')
def test_next_color_gets_next_in_list(self, mock_os, mock_mixer):
    player = Player()
    self.assertEqual(
        player.color,
        settings.COLORS[0],
        'The player should start out as the first color.'
    )
    player.next_color()
    self.assertEqual(

```

```

        player.color,
        settings.COLORS[1],
        'next_color should get the next color of all colors'
    )
    player.prev_color()
    self.assertEqual(
        player.color,
        settings.COLORS[0],
    )
    player.prev_color()
    self.assertEqual(
        player.color,
        settings.COLORS[-1],
        'prev color wraps around to the end.'
    )
    player.next_color()
    self.assertEqual(
        player.color,
        settings.COLORS[0],
    )

@mock.patch('lazer_blast.ships.pygame.mixer')
@mock.patch('lazer_blast.ships.os')
@mock.patch('lazer_blast.ships.pygame.draw.rect')
def test_when_laser_on_rect_drawn(self, mock_draw, mock_os, mock_mixer):
    player = Player()
    player.box = pygame.Rect(0, 0, 10, 10)
    player.surface = pygame.Surface((500, 500))
    player.render(player.surface)
    self.assertEqual(mock_draw.call_count, 0)
    player.flip_laser()
    self.assertTrue(player._laser)
    player.render(player.surface)
    self.assertEqual(
        mock_draw.call_count,
        1,
        'It should have called once for the player, '
        'And one more time for the laser shape. '
    )

class EnemyTestCase(unittest.TestCase):

    def setUp(self, surface=None):
        self.rand = random.Random()

    def test_enemy_moves_from_top_to_bottom(self):
        enemy = Enemy()

```

```

        enemy.surface = pygame.Surface((500, 500))
        enemy.box = pygame.Rect(0, 0, 10, 10)
        ticks = self.rand.randint(3, 10)
        for i in range(ticks):
            next(enemy)
        expected = ticks * settings.ENEMY_SPEED
        self.assertEqual(
            enemy.box.top, expected,
            'The enemy should have moved down the screen.'
        )

    def test_in_bounds_method_tests_if_corner_is_in_bounds(self):
        enemy = Enemy()
        enemy.surface = pygame.Surface((500, 500))
        enemy.box = pygame.Rect(10, 0, 10, 10)
        self.assertTrue(
            enemy.in_bounds(),
            'The upper edge is in bounds.'
        )
        enemy.move(0, 400)
        self.assertTrue(
            enemy.in_bounds(),
            'In the middle is in bounds.'
        )
        enemy.move(0, 99)
        self.assertTrue(
            enemy.in_bounds(),
            'On the bottom edge is in bounds'
        )
        enemy.move(0, 2)
        self.assertFalse(
            enemy.in_bounds(),
            'Past the bottom is out of bounds'
        )

    @mock.patch('lazer_blast.ships.Random')
    def test_enemy_may_move_side_to_side(self, mock_random):
        rand = mock.MagicMock()
        mock_random.return_value = rand
        rand.random.return_value = 0.99
        enemy = Enemy(starting_pos=(50, 0), color=(255, 0, 0))
        enemy.surface = pygame.Surface((100, 100))
        self.assertEqual(
            enemy.box.left,
            50
        )
        next(enemy)
        self.assertEqual(

```

```

        enemy.box.left,
        50 + settings.ENEMY_SPEED,
        'If the random number is within bounds of 1.0 turns right.',
    )
    rand.random.return_value = 0.01
    next(enemy)
    self.assertEqual(
        enemy.box.left,
        50,
        'If the random number is within bounds of 0.0, turns left.',
    )
    rand.random.return_value = 0.5
    next(enemy)
    self.assertEqual(
        enemy.box.left,
        50,
        'If the random number is within bounds of 0.5, goes straight',
    )

def test_enemy_takes_damage_when_hit(self):
    surface = pygame.Surface((100, 100))
    enemy = Enemy()
    enemy.surface = surface
    enemy._is_hit = lambda: True
    next(enemy)
    self.assertEqual(
        settings.ENEMY_HEALTH - settings.PLAYER_STRENGTH,
        enemy.health,
    )

class PlayerEnemyInteractionTestCase(unittest.TestCase):

    def setUp(self):
        color = (255, 0, 0)
        surface = pygame.Surface((100, 100))
        self.player = Player()
        self.player.surface = surface
        self.player.box = pygame.Rect((45, 90, 10, 10))
        self.player.color = color

        enemy1 = Enemy()
        enemy1.color = color
        enemy1.surface = surface
        enemy1.box = pygame.Rect((45, 0, 10, 10))
        enemy1.target = self.player
        enemy2 = Enemy()
        enemy2.color = color

```

```

    enemy2.surface = surface
    enemy2.box = pygame.Rect((55, 0, 10, 10))
    enemy2.target = self.player
    self.enemies = [enemy1, enemy2]

def test_player_knows_if_enemies_are_touching(self):
    enemy1, enemy2 = self.enemies
    self.assertFalse(
        self.player.enemies_touching(self.enemies),
        'No enemies should be touching the player.'
    )

    enemy1.box = pygame.Rect((50, 90, 10, 10))
    self.assertTrue(
        self.player.enemies_touching(self.enemies),
        'One enemy should be touching the player.',
    )
    enemy2.box = pygame.Rect((51, 90, 10, 10))
    self.assertTrue(
        self.player.enemies_touching(self.enemies),
        'Two enemies should be touching the player.',
    )

def test_enemy_takes_damage_from_lazer_only(self):
    self.player.flip_laser(True)
    enemy1, enemy2 = self.enemies

    enemy1.box.left = self.player.laser.left
    enemy1.box.top = self.player.laser.top + 20

    # half-way overlapping with player (below)
    enemy2.box.left = self.player.box.left
    enemy2.box.top = self.player.box.top + 5

    self.assertEqual(enemy1.health, settings.ENEMY_HEALTH)
    self.assertEqual(enemy2.health, settings.ENEMY_HEALTH)

    next(enemy1)
    next(enemy2)

    self.assertEqual(
        enemy1.health,
        settings.ENEMY_HEALTH - settings.PLAYER_STRENGTH
    )
    self.assertEqual(enemy2.health, settings.ENEMY_HEALTH)

def test_enemy_doesnt_take_damage_from_inactive_laser(self):
    self.player.flip_laser(False)

```



```
enemy = self.enemies[0]

enemy.box.left = self.player.laser.left
enemy.box.top = self.player.laser.top + 20

next(enemy)

self.assertEqual(enemy.health, settings.ENEMY_HEALTH)
```

Data files used if any

Sample runs

Users manual

Lazer Blast

“Lazer Blast” is an arcade style shooter (like [Galaga](#)), which includes aspects of color matching and pattern matching.

Running

To install and run the game, clone this repository and `cd` into it in a terminal. Then, run:

```
python3 setup.py install
```

You can then enter the command,

```
lazerblast
```

And the game will start.

Game-play

The user pilots a small ship against hoards of enemies. Each enemy has a given color. The player must use a similar-colored laser to destroy that enemy. For example, an enemy which is green much be shot with a green laser. A red laser will have no effect. Waves of enemies come in patterns: an astute fighter will be able to predict the next wave from the previous, and will have a significant advantage over the enemy.

Since the player cannot be expected to handle the copious masses of colorful enemies immediately, the difficulty of the game increases as the player progresses, starting with a single color and slowly adding other colors and increasing the speed.

Development

1. Clone the repository.
2. Install the requirements and game:

```
pip install -r requirements.txt
python3 setup.py develop
```

NB. In the future, the requirements should be handled by the setup file.

3. Run the game:

```
lazerblast
```

NB. Using a `virtualenv` would be preferable, but it does not work (at least, not on a mac.) See [this](#)

Running Tests

Tests are run with the `pytest` runner through the setup utility:

```
python3 setup.py test
```

Articles Collected and Used if any