2142 (SpaceBunny)
Summary and Design Report

Version <1.0>

Diego Franchi
Michael Romero

2018-05-08

Revision History

Date	Version	Description	Author
05/08/07	<1.0>	Initial Version of Document	Diego Franchi

Table of Contents

Introduction

Team Name	DMX
Team Members	Diego Franchi
	Michael Romero

What is your project?

Game Name	2142
Game Description	PyGame tribute to 1942 in space
Game Fiction	Space Bunny pilots a fighter ship to defend the earth from an alien invasion.

Game Genre/Sub-genre

Game Genre	2D Vertical Scroller
Game Sub-genre	Sci-Fi
	Shoot-em-Up
	Bullet Hell

Project Details

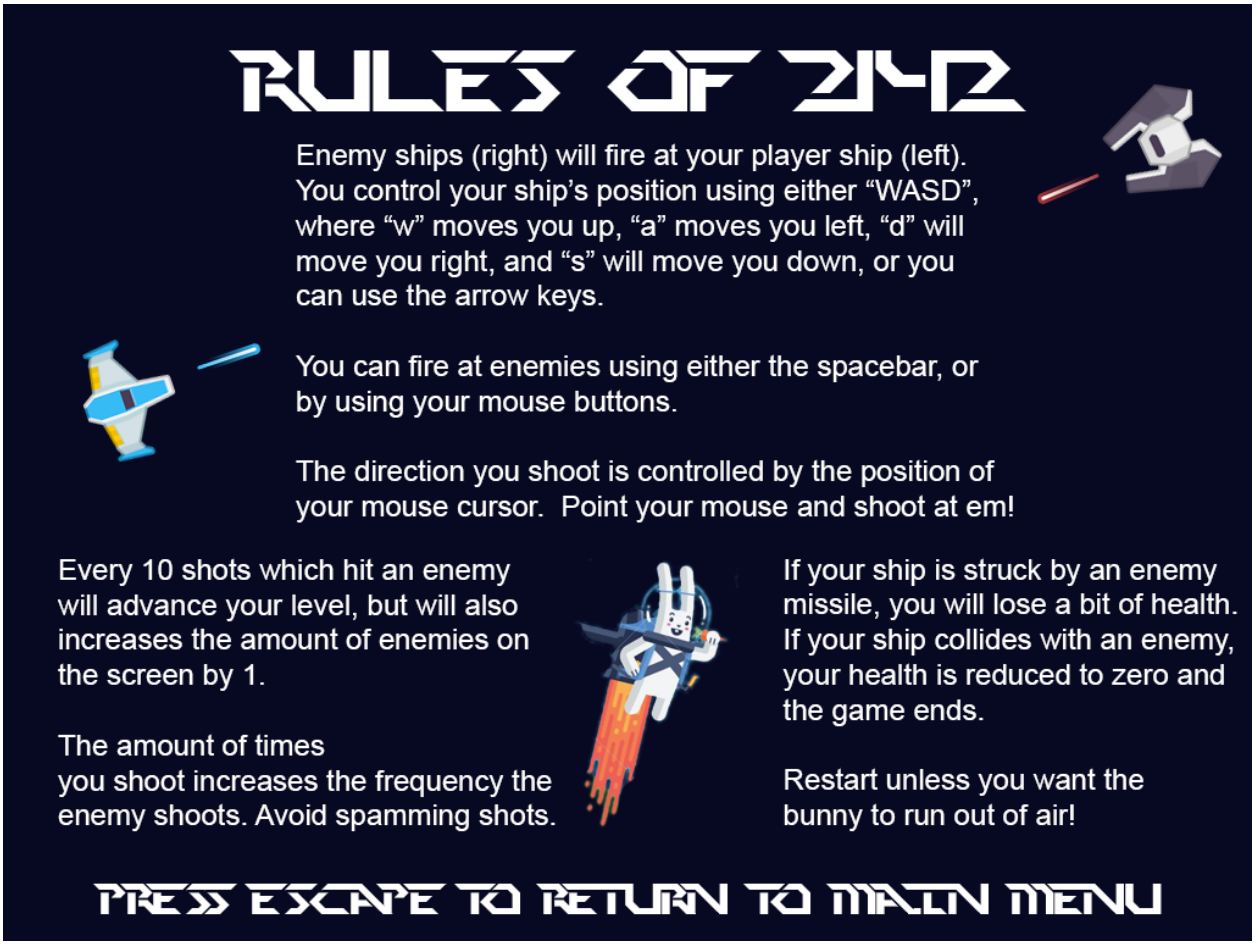
2142 is a space shooter reminiscent of 2D arcade scrollers from the 1980's. The objective of the game is to shoot enemy ships as they fly in from the top of the screen to get the highest score. You as the player travel through space with the ability to move freely in eight directions. The space ship is equipped with an updated aiming system that allows for precision targeting in a full 360 degrees with the mouse. The main action mechanic the player and enemies have are firing destructive laser cannons. Clicking the left mouse button fires a powerful laser that, on collision, will make enemies explode and give the player score. Watch out for enemy lasers as they damage the player. Take enough damage and game over. Eliminating enough enemies takes you to the next level where the difficulty increases as more enemies begin to spawn. Compete against your friends and see who can get the highest score!

Project Utility

The constant flow of enemies challenges the player to weave through a bullet storm to find the best angles and land those satisfyingly perfect shots.

Design

Rules + Controls



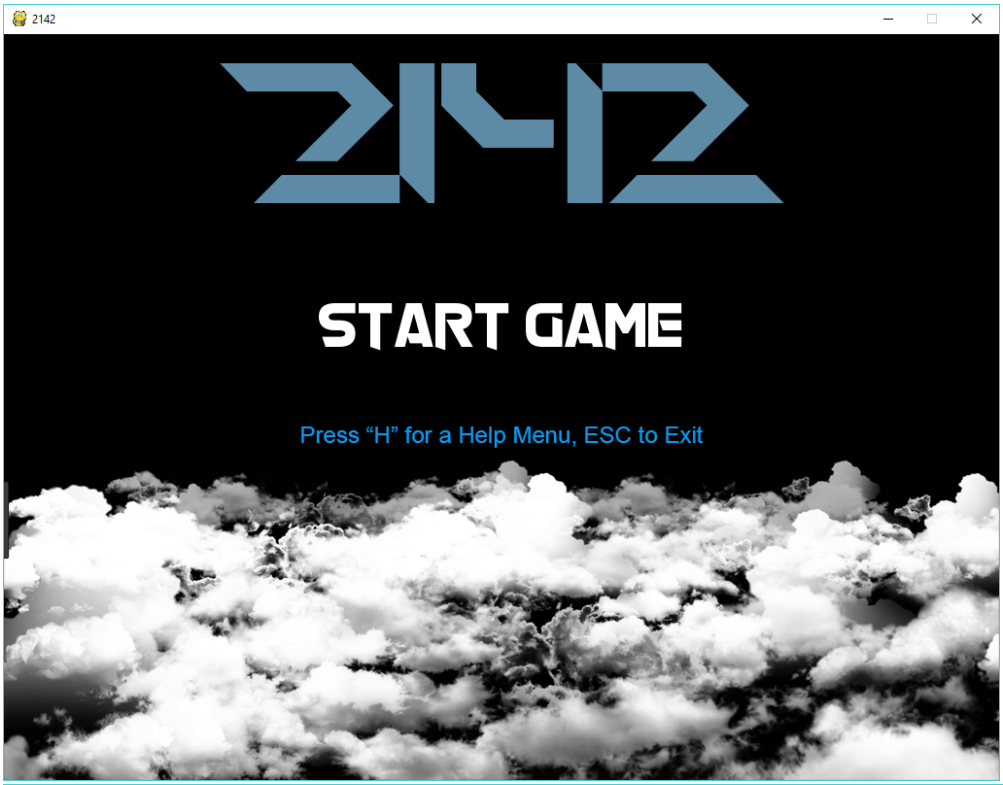
Scoring

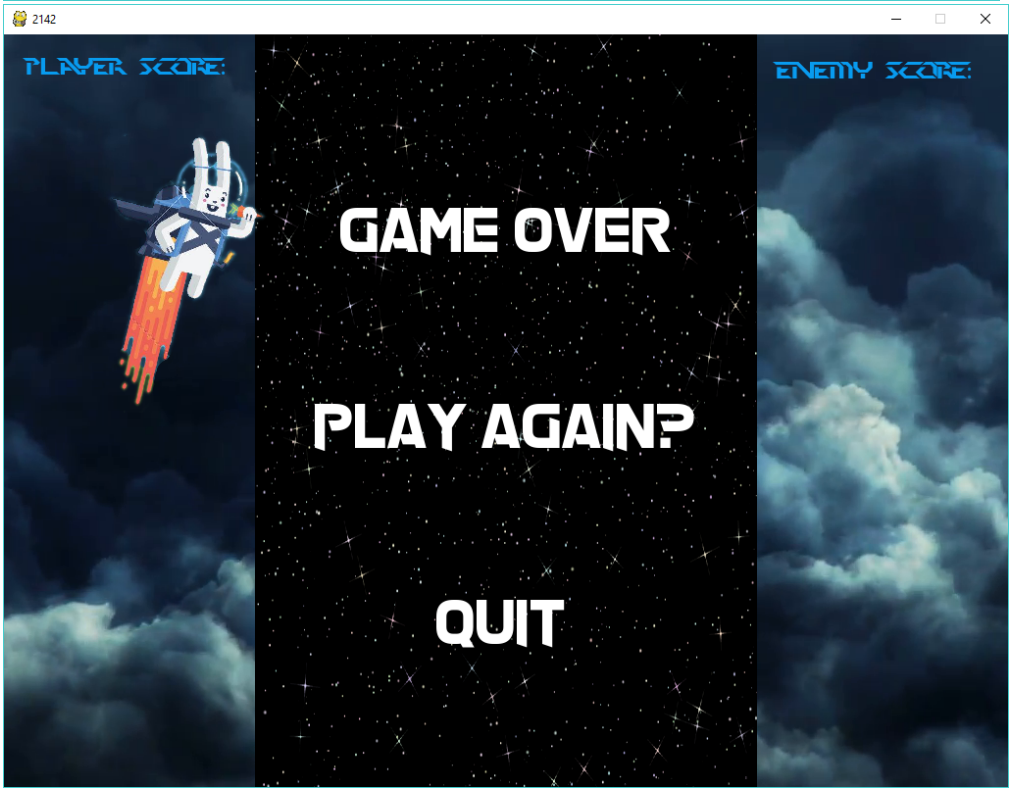
High score is calculated by enemies eliminated. 1 score point = 1 enemy slain. Level progression is also calculated by the current player score with every 10 points advancing the player to the next level and increasing the number of enemies on the screen by 1. Enemies have a score as well calculated by the number of times they hit the player.

Expected Duration of a Game

The expected duration of a game is less than a minute. This is due to the frequency at which enemies spawn, the frequency enemies fire lasers, and the rate at which the player progresses through the levels to increase the formerly mentioned attributes.

User Interface

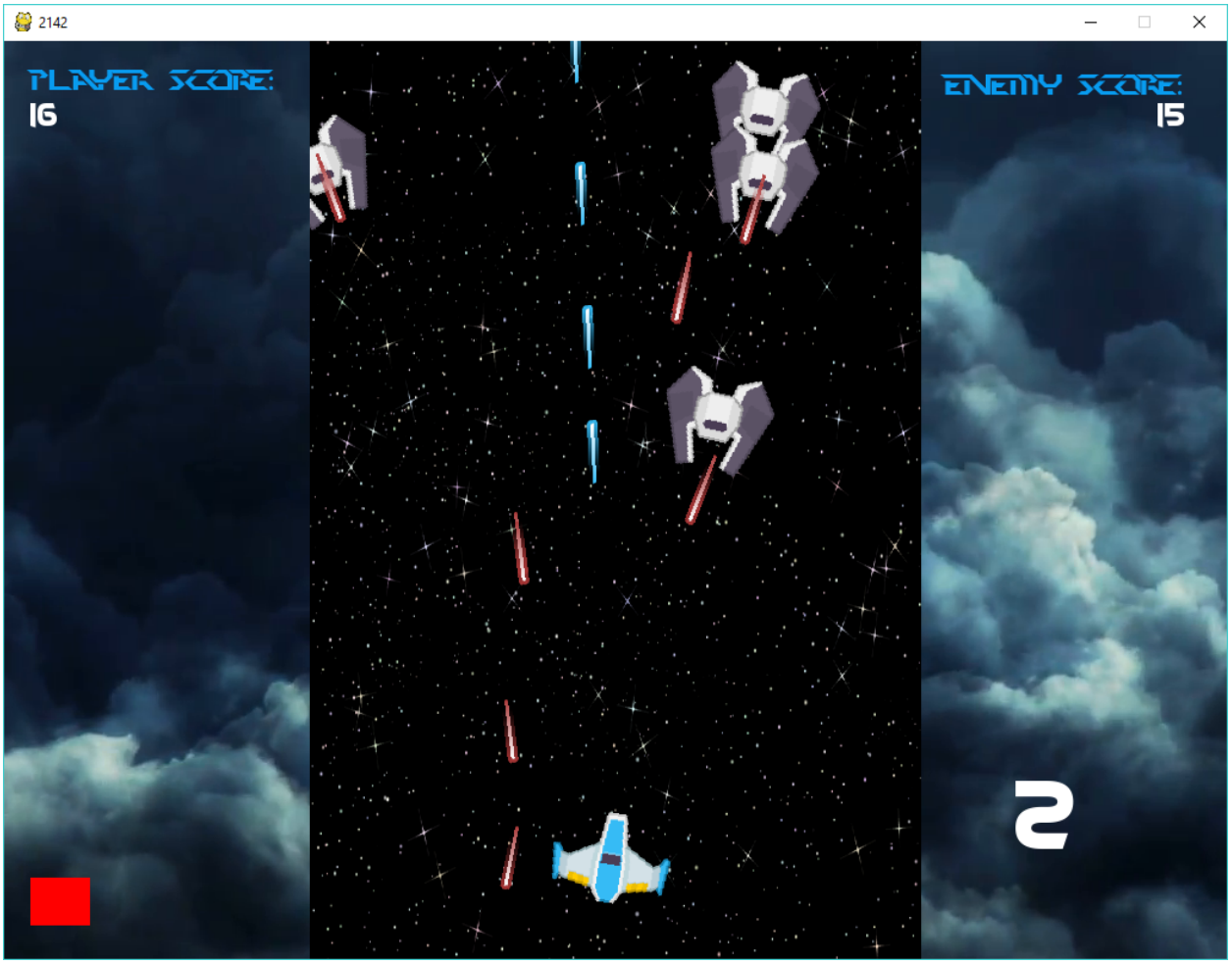




Win State, Lose State

2142 does not have a traditional win state. Instead, he player plays until their health total reaches zero (lose state) to see how far they can get in the game.

Visual Representation of Game State



Expected Skills of the Player

2142 rewards players with fast reaction time (to dodge incoming laser attacks) and fast clicking skills (to eliminate as many enemies as fast as possible). The barrier of entry is very low, but by no means is this a casual game.

Sources of Uncertainty

The game is an endless challenge. We as the developers are uncertain of the highest level possible to be reached by a player. Our best performer reached level 14 before being defeated. A solution would be to add a challenging boss level after a certain point to close out the game.

Software Architecture

Algorithms Used

def evaluate_menu_click (event, menu_buttons)

Algorithm used to evaluate whether a mouse click event is in the bounding box of a button. Input: pygame.event object to determine location of the click; list of menu buttons on screen. Output: None if no button clicked; otherwise the button clicked

def game_menu ()

Algorithm used to create the main menu of the game programmatically. Plays menu music on an indefinite loop, displays input for a rules sheet and displays a button to begin the main game. Output: True on ENTER input or if user clicks start game button; False on ESC input or if user closes the window

def new_game ()

Algorithm used to reset all initial values to their starting values. Example: level = 1, score = 0

def draw_game

Algorithm used to scroll the background to simulate space travel, display player score, enemy score and health total. Also uses class functions and game objects to determine the direction and speed a player or enemy is moving to accurately draw the next game state.

def game_loop ()

Algorithm used to process all player inputs, play game music, calculate the current level based on the score and check if the lost state has been satisfied to draw the game over screen. This is also where the games play space is bounded and where bullet collision is calculated. draw_game() and game_over() are called here.

def game_over ()

Algorithm used to display a menu for the player to start another game or return to the main menu. Space Bunny bounces around on the screen until the next mission. Output: True if play again is selected; False if quit is selected (this output value is returned to game_loop())

def main ()

Calls the game_menu and depending on the return value will start a game_loop().

Software Organization

The whole game is written in one python file called main.py. All game assets are organized into file folders called: fonts, images and sounds. The following

Imports

At the top of the document we import necessary modules to run our game. Importing pygame allows us to use pygame events to detect key presses and sequence the animation of our game. Importing random is used to generate the random positions enemies will spawn from at the top of the screen. Import math is used in all the trigonometric rotation calculations and used in calculating the vector a laser will be fired in.

Constants

Constant values are utility variables used throughout the application. These values include: GAME_TITLE, DISPLAY_WIDTH, DISPLAY_HEIGHT and several RGB color variables. Constants are never changed after their initial creation.

Initialization Values

Initialization values are used during the game loop. These global values describe the state of the game and thus are manipulated during gameplay. Some examples are: player_score and active_bullets.

File I/O

Many image and sound files are loaded at the beginning of the program. This is to frontload most of the file i/o to optimize the runtime algorithms since they only must be loaded once. Files included in this section range from fonts and sounds to images and backgrounds.

Object Oriented Programming

Every entity seen in the game is represented by an object-oriented class in the code. This aided tremendously in the initial design of the program and made programming game mechanics intuitive to understand as we created an interactive object system.

Conventional Python Style

Our python document uses conventional python practices in naming variables, white space usage, comments and code lay-out. The consistency of the code makes improves readability and makes it consistent across the wide spectrum of Python code.

Game Class/Objects

Enemy	
• set_location • next_location • rotate • fire	• EnemyBullet

Enemy describes the enemies in the game. Within the class are data members representing the enemy: image, health, speed, location and rotation. Major methods of an Enemy such as set_location, next_location and rotate are used to position the enemies on the screen. Next_location is calculated with the speed and direction set when the object is created. Rotation always points enemies toward the player. The Enemy class collaborates with the EnemyBullet class as enemy ships in game fire bullets with different properties than the player bullet.

EnemyBullet	
• set_location • next_location • rotate	• Enemy

EnemyBullet object represents the lasers fired by enemies. Their image is passed through the class constructor and its main functions serve the same purpose as enemy set_location, next_location and rotate do for animation.

Player	
• set_location • rotate • fire • up • down • left • right • up_left • up_right • down_left • down_right	• Bullet

Player objects have a similar design as enemies with the exception that players can control their movement therefore not needing a next_location function. Players can freely move in eight directions defined in the functions up, down, left, right, up_left, up_right, down_left and down_right. These functions work by changing the x and y coordinates of the player when corresponding input keys are pushed. The rotation function in the Player class will always direct it to the coordinate position of the mouse in the game. This along with the fire function being bound to mouse click makes the game much faster paced than in initial iterations of the code. The Player class collaborates with the Bullet class.

Bullet	
• set_location • next_location • rotate	• Player

Bullet objects represent the lasers fired by the Player. Set_location, next_location and rotate functions are used during the game_loop() to animate the firing an collision of bullets against enemies.

Game Demonstration

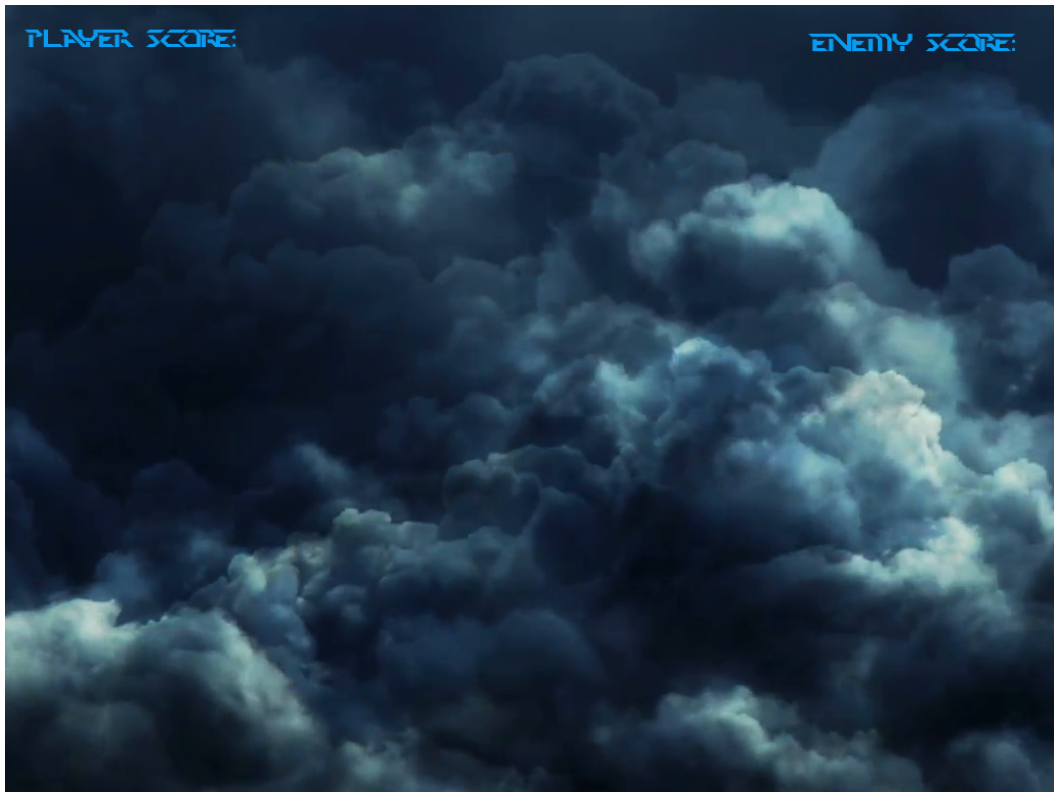
2142 (SpaceBunny)	Version: <1.0>
Summary and Design Report	Date: 2018-05-08
386-p3_DMX	

1. Player Score (every 10 player points increases the level)
2. Enemies spawn at the top of the screen and fire lasers at the player as they travel downward
3. Enemy Score (currently used only to “play” against the enemies)
4. Player can dodge enemy lasers and fire back to earn score
5. Health Bar (when reaches zero results in player game over)
6. Level Indicator (every level increases the number of enemies on screen by 1)
7. Foreground image scrolls downward to simulate space travel

Bibliography

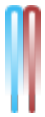
Copyright Disclaimer Under Section 107 of the Copyright Act 1976, allowance is made for "fair use" for purposes such as criticism, comment, news reporting, teaching, scholarship, and research. Fair use is a use permitted by copyright statute that might otherwise be infringing. Non-profit, educational or personal use tips the balance in favor of fair use.

Source: https://www.videoblocks.com/video/seamless-3d-animation-of-aerial-view-of-cloudy-sky-with-clouds-with-camera-moving-in-alpha-transparent-background-in-4k-loop-b94rwm7titgbgjbv



Source: https://www.videoblocks.com/video/storm-clouds-background-loop-animation-s4ayivyhit5rolye

Source: http://www.sky-map.org/



Source: http://kenney.nl/assets/space-shooter-redux

Source: https://thumbs.dreamstime.com/z/startup-busines-development-project-concept-progress-achievement-flying-rocket-jetpack-rabbit-launching-sky-over-ocean-90332637.jpg

Off The Haze.otf

Source: https://www.abfont.com/off-the-haze.font

weapon_player.wav

weapon_enemy.wav

sfx_laser2.ogg

oakenfold.ogg

music_background.wav

menu_audio_01.ogg

explosion_player.wav

explosion_enemy.wav

explosion_asteroid.wav

Source: https://freesound.org/