
Python version: 3.7

LEGIT BIT BLITZ by Calvin Wong and Nabeel Hingun

INSTRUCTIONS (SINGLE PLAYER ELIMINATION):

- WASD to move.
- Mouse to aim.
- Mouse Click to shoot (you can hold).
- Shift to sprint.
- Press 1 to switch to shield, which can bash enemies
- Press 2 to switch to machine gun,
- Press 3 to switch to shotgun,
- Press Tab to cycle through weapons
- Shoot all enemies without getting shot; your lives are in the top right, and the enemy count is at the bottom right.
- If you have hardmode on, shoot all the enemies, but keep an eye out when you defeat all the regular enemies (once it comes, something might appear in the top left corner too)...

INSTRUCTIONS (TWO PLAYER):

Player 1:

- WASD to move.
- T and Y to aim.
- Automatically shoot (No need to manually do it).

Player 2:

- Arrow keys to move.
- Right shift and Enter to aim.
- Automatically shoot (No need to manually do it).

- Shoot the other player. You each have 5 lives. Each is listed in the top corners.

FEATURES:

1.) *Main menu*

We have a main menu which prompts the user to choose which game mode he wants to play. Both game modes are arcade shooters. In the first one, the player moves his unit and can shoot enemies. Before starting the game, the user can choose a number of parameters such as the number of enemy units, the number of friendly units, the speed of the player and the level of difficulty. The second game mode is a 1 v 1 game where two users can face each other. After the main menu, there also is a intro before the game starts.

2.) *Spawning enemies*

We have a class, which is similar to a template or blueprint for something, and that class is called ‘enemy’. These enemies are generated according to the class traits and functions, basically modeled after the blueprint, and then we use a function from the random module to generate random x and y coordinates in a certain range such that the enemies spawn at the top (ie: x coordinates are within the game “walls.” There are 4 walls that make up the game’s boundaries; if a player or unit reaches the wall, they bounce or stop. The y coordinates are between around the middle of the screen and the top half of the game boundaries).

3.) *Moving units*

We used class traits to set a x and y speed for each unit, then an update function for units that added the x and y speed to the respective x and y coordinates.

To bounce units off of barricades and walls, if a collision was detected between the unit and the wall, if the x speed was a certain value it was reversed and the same with the y speed.

4.) *Firing*

There are different units which can fire. It can be the player, the enemy units or the friendly units. We also have a class which we called ‘bit’ which contains the variables and functions for the bullet. For a unit to fire, it has to call on the “bit”

class so that it can spawn a bullet and so that the bullets have the properties and functions of the “bit” class.

LISTS AND DICTIONARIES USED:

List/Dictionary Names (Which_Py_File_It’s_In)	**Usage:**
SPLATTER_LIST (Setup.py) Line 373	We used a for loop to load a series of images and append them to SPLATTER_LIST. Items from this list were then used to animate the Splatter class.
PAIN_SOUNDS (Setup.py) Line 380	We used a for loop to load a series of sounds and append them to PAIN_SOUNDS. When a unit lost hp, a random item from this list was played.
MUSIC (Setup.py) Line 393	MUSIC was created using a list comprehension, and is a list of indices that was used to randomly choose the menu music from 3 files.
“collision_group” function in conjunction with the “hit_detection” function (CalvinWong_NabeelHingun_LegitBitBlitz.py) Line 1247, 1258, 1268, 1271, 1272	Collision_group creates a dictionary with keys being sprites from 1st group and values being a list of all sprites from 2nd group key sprite collided with; hit_detection operates on those dictionaries, such as in the enemy_hit block of code or the friendly_hit block
“Calc_shoot_vect_ang” function (in the Unit Class) (CalvinWong_NabeelHingun_LegitBitBlitz.py) Line 110	Uses a list called “targets” to store potential positions to shoot at, iterates through a pygame Group data structure (which we could not use random.choice on) and adds each sprite to the targets list, which then allows us to use random.choice to select a random target’s position to shoot at

---

LOCAL VARIABLES USED:

Function (Which_Py_File_It's_In)	**Local Variable(s) Used**	**Description of Usage**
choose_bit_speed function (Setup.py) Line 50	do_intro	"do_intro" is a local variable that determines if the top text has already been said.
Return_bool_probability (Setup.py) Line 249	power, upper_bound, max_outcome, range_list, choose	"Power" stores the amount of decimals the probability has, "upper_bound" is a mathematical variable used to calculate "max_outcome", the highest integer that allows the outcome to occur for a probability simulation. "Range_list" is the list of integers from 0 to the upper bound, and then the simulation chooses and random number, "choose", which then is checked to see if the random number is less than the upper bound, in which case the simulation returns that the event did occur.
All of our classes	self.variables are all instance variables unique to those instances, only accessible by that instance.	Used to store traits and data about each class and their instances
Most of the spawn functions (eg: spawn_fb, spawn_unit) (Line 1145)	Use a temporary variable (eg: fb - line 1472, unit - line 1148) to spawn and	Used so that you don't have to store each instance with their own

	store an instance of a class, append it to a pygame Group, then spawn in a new one and store it in that temporary variable as needed	unique name, merely have to add it to the pygame Group

---

PYTHON PACKAGES USED:

Note: Only pygame needs to be installed, the rest are pre-installed with python.

1. Pygame
   https://www.pygame.org/news
   To install, use “pip install pygame” in the command prompt

2. Os
   https://docs.python.org/3/library/os.html

3. Math
   https://docs.python.org/3/library/math.html

4. Random
   https://docs.python.org/3.7/library/random.html

ADDITIONAL INFORMATION / REQUIREMENTS:

Make sure the graphics files (.png) are in a folder called “GFX” in the game directory and the audio files (.ogg, .wav) are in a folder called “AUDIO” also in the directory. “CalvinWong_NabeelHingun_LegitBitBlitz.py”, “Setup.py”, and “Variables.py” should all be in the game directory.

Also, why we should get *extra credit*:

TLDR:

- **The sheer amount of effort put into this project**
- **Time sink of creating an Original Game, learning Pygame, Python OOP, Os, time, and random modules**
- **Manipulating pygame data structures**
- **Incorporated Recursion (determine_decimal_places function, spawn_units_recursive function, lots of the choose a variable functions)**
- **Math Heavy for rotating sprites and calculating how to shoot at a position**
- **Two Game Modes and able to set variables in the console that catch errors using “try/except” and if statements,**

We wrote a new game, rather than basing it on an old game like Plants v.s. Zombies or Tetris. This meant we had to plan it all from scratch, which was a huge time sink.

One of the first super hard roadblocks was manipulating lists and data structures with Pygame’s group function. Pygame groups are unhashable and thus can't be indexed, but yet they allow duplicate items and have the mutability of lists; it was weird manipulating it, especially when trying to figure out how to interact with only certain sprites upon collisions. We came up with our own original solution in the “Calc_shoot_vect_ang” function (in the Unit Class in CalvinWong_NabeelHingun_LegitBitBlitz.py) using the targets list in line 110, essentially adding the sprites that collided to a new list that could be indexed, and thus could be operated on by the random.choice() function.

There also some math we had to figure out for the game physics. For example, we figured out a statistical function to simulate a random chance generator in line 249, the “return_bool_probability” function in Setup.py, which itself required us to figure out a function that determines the decimal places of a number. We also figured figured out how to use arctangent to calculate the vectors necessary to shoot a bit in any direction at a certain speed, and to calculate what angle the bit should be rotated. Again, the pygame reverse y coordinates threw us off, so that was quite a time sink, especially when trying to rotate a sprite.

We learned how to use the super function for not overwriting parent functions,
And try/except statements to allow for users to input the correct input (and have some snarky comments :])

Moreover, we also have two game modes, rather than a single game, and for such a large project, the code is relatively organized.

We learned about pygame time functions, using that to do functions at certain times, and there was the audio and image systems we had to learn. We learned how to set directories using the os module, and overall we incorporated a lot of material into this game.

The vast majority of our program used OOP in python with the game itself being a class. It was difficult at first because OOP in Snap and in Python are different. Here, there’s classes, self, inheritance, the super function, and dot notation that takes a while to learn and use effectively.

With Pygame, we had to learn about rects, sprites, surfaces, the pygame coordinate system (which is trippy since the y axis is reversed, and we had a camera system that didn’t move the screen; rather, it moved the objects, so we had to calculate relative coordinates rather than having constant coordinates),

---

TABLE OF CLASSES / FUNCTIONS:

NOTE: CLASSES are highlighted in BLUE

Also please note that all classes have “__init__” and most have “update” functions. We did not include every single one if them in the table.

The “__init__” function determines what the class should do/what traits (variables) it when it is initiated (whenever the instance of the class is created).

The “update” function either changes some of the attributes of the instance or calls upon the methods of the class every time you call it; pygame has a special “Group” data structure that allows you to call the update function en masse.

Block/Function Name	**Domain(Inputs)**	**Range(outputs)**	**Behavior (what exactly a given block/function does in the context of your project)**
startup	none	none	Initializes settings for pygame and sets the title of game window
choose_resolution	none	none	Asks user for dimensions of game window
choose_bit_speed	boolean	none	Asks user to choose speed at which units move in the game
choose_player_speed	boolean	none	Asks user to choose speed at which player moves
ask_number_of	String, boolean	Number (integer)	Asks the number of “string” the user wants in the game, and returns the input if it is an integer
choose_hardmode	boolean	none	Asks the user if they want to activate hardmode, and does so if the

			answer is 1, or not if the answer is 2; otherwise it asks again.
choose_gamemode	boolean	none	Asks the user which gamemode (singleplayer and two player) they want to play, or if they want a description of each gamemode.
replay	none	none	Asks the player whether he/she wants to restart the game (not used because of issues with pygame.quit())
load_music	string	none	Stops the current music (if any), then loads and plays a music file with the inputted name
load_sound	string	none	Loads the sound file with the inputted filename
random_direction	none	Number (integer)	Returns either 1 or -1 (randomly chosen)
pick_one	Any value type	Any value type	Chooses a variable randomly from 2 variables with equal probability
determine_decimal_places	Number (integer)	Number (integer)	Recursive function that tells how many decimal places the number has
return_bool_probab	Number	Boolean	Takes in a

ility			probability and does a probability simulation that returns true if the simulation returns the outcome (eg: if you input .33, which means it has a 33% chance of occuring, it'll generate a range of numbers from 0 to 99, and then choose a random number. If the random number is less than 33, then it returns True.)
say_centered	String, number, tuple (RGB color)	None	Takes in a string and draws it on the screen in the center with the size and color inputted
say	String, number, tuple (RGB color), number, number	None	Takes in a string and draws it on the screen at the inputted x and y coordinates with the size and color inputted
collision_group	Pygame Group, pygame Group, boolean, boolean	Dictionary	Shorthand for the pygame collision detection algorithm; detects collision between group 1 and group 2, with del_group being a bool telling the computer whether to delete that instance when it

			collides
WaitTimer	Base class	Instance	Was used as a demonstration of our understanding of pygame's clock; useful because of the function below
wait_over	Instance, integer	Boolean	Takes in a time in milliseconds and returns True if the time since the last time it returned True is greater than or equal to its the inputted time. Otherwise, it returns False.
FallingBlock	Base class	Instance	A square particle that falls down and reappears at the opposite side if it falls out of the screen
random_speed	Instance, none	none	returns a random speed within a certain ratio of the original speed
SpawnPoint	Base class	Instance	Creates a sprite that stores the spawn point of the boss (since the camera algorithm changes everythings' position)
Unit	Base class	Instance	The base class for the Player and its allies and enemies, determines basic

			traits like firing and moving
change_size	Instance, integer	none	Changes the dimensions of the unit but keeps the center the same
move_one_direction	Instance, number, number	none	Moves the unit in one direction
move	Instance, number, number	none	Uses the previous function to move the unit in both directions
reset_direction	Instance, String	none	Sets speed in one axis to a slower or faster speed
bounce	Instance, string	none	Makes the unit bounce if it hits a barricade
calc_shoot_vect_ang	Instance	Numbers (3)	Calculates the x-direction, y-direction and angle of the bullet for shooting at a random target
which_side	Instance, number, number	Tuple (coordinates)	Determines which side of the unit the bullet will spawn on
shoot_single	Instance	None	Shoots a single bit
shoot_shotgun	Instance	None	Shoots a spread of bits
shoot_first_shot	Instance	None	Used to shoot the first shot and set an attribute such that not every unit fires at once

fire	Instance	None	After first shot, fires every x seconds where x is determined by the unit type
Enemy	Base class	Instance	Units that shoot at friendlies and the player
change_size	Instance, number	None	Changes the enemy instance's size
Boss	Base class	Instance	The final boss that spawns in hardmode if you've shot all the enemies, switches from shotgun to minigun and targets random friendlies
Friendly	Base class	Instance	Units that shoot at enemies and the boss
Player	Base class	Instance	The unit that the user controls to move and shoot
stop	Instance	none	Stops the player if it hits a barricade
calc_shoot_vect_ang	Instance	Numbers (3)	Calculates the x-direction, y-direction and angle of the bullet for shooting at the mouse
del_shield	Instance	none	Destroys the shield of the player
Bit	Base class	Instance	The bullets of the game, customized

			based on what team is inputted; draws and rotates itself based on the inputted angles and vectors
hit	Instance	none	Decreases the hp of the bullet and destroys it if it has 0 hp left
Wall	Base class	Instance	Serve as the boundaries that units can't move past
Barricade	Base class	Instance	Rectangles that units can take cover behind, shields against incoming bits
PlayerShield	Base class	Instance	Shield that blocks bullets, controlled by and aimed at the mouse
Splatter	Base class	Instance	Animated sprite that serves as the electric splatter that comes when a unit is hit
create_screenshake	Instance	none	Allows the Splatter class to use a camera algorithm to mimic a screen shake
extend	Instance	none	Extends the duration of the screenshake
screen_shake	Instance	none	Moves all sprites to give an impression

			that the screen is shaking
Player2P	Base class		Player in 2nd game mode
shoot	Instance	none	Creates an instance of the Bit class
Aim	Base class		Sprite which rotates about the player
Bit2P	Base class		The bit class used in two player mode, allows for bouncing off barricades and off screen “teleportation”
bounce	Instance	none	Reverses direction of bullet
Barricade2P	Base class	none	Walls in the 2nd game mode
Game	Base class	none	Hosts functions and variables so that functions can be repeated easily, allowing us to transition between the menu, the game, game over screens, etc, as well as to create game state variables that don't need to be global variables
game_quit	Instance	none	Set some global variables to False to stop the game loop

press_esc	Instance	none	Set some global variables to False to quit the game loop
quit_loop	Instance, event	none	Checks if user presses key to quit game
spawn_fb	Instance		Spawns a falling block
main_menu	Instance	none	Displays name of game and instruction on screen
spawn_unit	Instance, class name	none	spawn units of the "unit_class" type, and append them to the correct pygame Groups, and edit the unit counters
spawn_units_recursive	Instance, number, class name	none	Uses recursion to spawn units
spawn_barricade	Instance, number	none	Spawn a singular barricade at (x, y)
spawn_all_barricades	Instance, number	none	Uses list comprehensions to create a "grid" of coordinates where the barricades are spawned at
spawn_border	Instance	None	Spawns in a box of walls
player_collide	Instance	none	Creates a Splatter, a screen shake, decreases player hp by 1, and

			checks if player hp has reached 0 or if they've bashed all enemies to death
hit_detection	Instance	none	Checks detection between the different groups of sprites and deleted the instance of the class
prep_phase	Instance	none	Gives instructions to user
spawn_boss_spawnpoint	Instance	none	Spawns the boss's spawnpoint sprite
spawn_boss	Instance	none	Spawns the last enemy: the boss
battle_phase_elimination	Instance	none	The loop that handles the single player mode
win	Instance	none	Displays a win message on the screen for user and plays applause sound
check_for_winner_2_p	Instance	boolean	Ends game if any player has won and prints which player has won
hit_detection_2_p	Instance	none	Detects hits between the bullets and the players and substracts hp from players if they are hit
spawn_barricades_2_p	Instance, Number (integer)	none	Calls on instances of the Barricade2P class and spawns

			them at random locations
battle_phase_2_player_duel	Instance	Display on the screen	Runs most functions to fire bullets, detect hits, print the hp of each player, draw the sprites
win_2_p	Instance, string	string	Says which player won
check_respawn_2_p	Instance, Event (from pygame)	none	Resets the hp and position for the two players
game_loop	none	none	Executes the game

Omitted several functions because the commented code should be sufficient.

Credits to:

Python creator

KidsCanCode for their Schmup Youtube Tutorials
https://www.youtube.com/watch?v=VO8rTszcW4s&list=PLsk-HSGFjnaH5yghzu7PcOzm9NhsW0Urw is the first lesson

https://www.reddit.com/r/pygame/comments/5r75q1/how_to_get_each_sprite_to_fire_a_bullet/ for help on time delay

https://www.pygame.org/project-Rect+Collision+Response-1061-.html for help on movement based on collisions

https://stackoverflow.com/questions/36510795/rotating-a-rectangle-not-image-in-pygame for help on creating a rotatable rectangle

---

Sound:

---

http://soundbible.com/2095-Mossberg-500-Pump-Shotgun.html
http://soundbible.com/1804-M4A1-Single.html
http://soundbible.com/1454-Pain.html
http://soundbible.com/947-Metal-Bang.html
http://soundbible.com/991-Left-Hook.html
http://soundbible.com/993-Upper-Cut.html
http://soundbible.com/462-Male-Grunt.html
http://soundbible.com/2015-Thunder-Strike-1.html
http://soundbible.com/1920-Minigun.html
http://soundbible.com/2021-Atchisson-Assault-Shotgun.html
http://soundbible.com/1260-Auditorium-Applause.html

Music:

---

https://www.dl-sounds.com/royalty-free/pim-poy-pocket/
https://www.dl-sounds.com/royalty-free/superboy/
https://www.dl-sounds.com/royalty-free/off-limits/
https://www.dl-sounds.com/royalty-free/power-bots-loop/
https://www.dl-sounds.com/royalty-free/defense-line/

And, so we can emphasize their importance by placing them as the best for last, the CS10 Fall 2018 Team. Thanks for the wonderful lessons on computer science. This really wouldn't have been possible without you guys.