Beaver Shooter Game – Code Documentation

Arcade game made by: Prestislav Dimitrov, Martina Aleksieva, Kristiyan Pavlov

This document provides a comprehensive overview of the "Beaver Shooter Game" Python project, detailing the purpose and functionality of each file and its constituent classes, methods, and important variables.

**1. main.py**

This is the entry point of the application. It initializes Pygame, sets up the game window, creates an instance of the GameManager, and runs the main game loop.

**Key Components:**

* **pygame.init()**: Initializes all the Pygame modules required for the game.
* **screen = pygame.display.set\_mode((WIDTH, HEIGHT))**: Sets up the display window with dimensions defined in constants.py.
* **pygame.display.set\_caption("Beaver Shooter Game")**: Sets the title of the game window.
* **game = GameManager()**: Creates the main game manager object, which orchestrates all game logic.
* **setup()**: A placeholder function, currently empty, but can be used for any one-time setup logic before the game loop starts.
* **update\_loop()**:
  + Handles user input events via game.handle\_events().
  + Updates the game state (beaver, bullets, enemies, collisions) via game.update().
  + Draws all game elements to the screen via game.draw(screen).
  + Updates the full display surface to the screen via pygame.display.flip().
  + Returns False if the game should quit, True otherwise.
* **main() (async function)**:
  + The asynchronous main loop that calls setup() once and then repeatedly calls update\_loop().
  + await asyncio.sleep(1.0 / 60): Controls the frame rate to approximately 60 frames per second.
* **Platform-specific execution**: Checks platform.system() == "Emscripten" to determine if it's running in a web environment (using asyncio.ensure\_future) or as a standalone Python application (using asyncio.run).

**2. constants.py**

This file defines global constants used throughout the game, including screen dimensions, colors, game mechanics parameters, and dynamic UI boundaries.

**Key Variables:**

* **WIDTH, HEIGHT**: Integers representing the width and height of the game window in pixels.
* **WHITE, BLACK, RED, BROWN, GREEN, CAP\_COLOR, BLUE**: Tuples representing RGB color values for various game elements and text.
* **MAX\_AMMO**: Integer, the maximum number of bullets the beaver can hold.
* **RELOAD\_TIME\_SECONDS**: Integer, the time in seconds it takes for the beaver to reload its ammo.
* **BULLET\_DELAY**: Integer, the minimum delay in milliseconds between consecutive shots.
* **STAGE\_ONE\_SPEED, STAGE\_TWO\_SPEED**: Integers representing score thresholds that determine the difficulty stage and enemy speed.
* **ENEMY\_MIN\_Y**: Integer, a dynamically updated global variable representing the minimum Y-coordinate (highest point on screen) where enemies can spawn and move. This is set by GameManager to be below the HUD.
* **BEAVER\_MAX\_Y\_UPPER**: Integer, a dynamically updated global variable representing the maximum Y-coordinate (highest point on screen) the beaver can reach. This is also set by GameManager to be below the HUD.

**3. beaver.py**

This file defines the Beaver class, representing the player's character, including its movement, shooting, and health.

**Class: Beaver**

* **\_\_init\_\_(self)**:
  + Initializes the beaver's width, height, x (fixed horizontal position), and y (initial vertical position at the bottom of the playable area).
  + Sets hp (health points), angle (gun angle), and speed (vertical movement speed).
  + Creates self.rect for collision detection.
  + Initializes max\_ammo, current\_ammo, reloading status, reload\_timer, and last\_shot\_time for shooting mechanics.
  + Loads and scales the beaver image (bobar\_1.png) and gun image (pushka.png).
  + Loads the ricochet\_sound for shooting.
  + Initializes self.gun\_tip as a placeholder, which will be accurately calculated in draw().
* **update(self)**:
  + Handles player input for gun aiming (K\_UP, K\_DOWN for angle) and vertical movement (K\_w for up, K\_s for down).
  + **Vertical Movement Constraints:** Uses max() and min() with constants.BEAVER\_MAX\_Y\_UPPER and constants.HEIGHT - self.height to ensure the beaver stays within the defined vertical playable area, preventing it from going into the HUD or off the bottom of the screen.
  + Updates self.rect.y to match the beaver's new y position.
* **draw(self, screen)**:
  + Draws the beaver image on the screen.
  + Calculates the position and rotation of the gun based on the beaver's position and current angle.
  + Draws the rotated gun image.
  + Calls \_calculate\_gun\_tip() to update the precise position of the gun's tip for bullet spawning.
* **\_calculate\_gun\_tip(self, gun\_x, gun\_y)**:
  + A helper method that calculates the exact coordinates of the gun's barrel tip after rotation. This is crucial for spawning bullets from the correct location.
  + It determines a pivot point and then applies trigonometric functions based on the angle to find the rotated tip coordinates.
* **shoot(self)**:
  + Checks if the beaver has ammo, is not reloading, and if enough time has passed since the last shot (controlled by constants.BULLET\_DELAY).
  + If conditions are met, it creates a Bullet instance at the gun\_tip position with the current gun angle.
  + Decreases current\_ammo and updates last\_shot\_time.
  + Plays the ricochet\_sound.
  + If ammo runs out, it sets reloading to True and records the reload\_timer.
  + Returns the new Bullet object or None if unable to shoot.

**4. bullet.py**

This file defines the Bullet class, representing projectiles fired by the beaver.

**Class: Bullet**

* **\_\_init\_\_(self, x, y, angle)**:
  + Initializes the bullet's starting x and y coordinates, speed, angle, and radius.
  + Creates self.rect for collision detection.
* **update(self)**:
  + Calculates the bullet's new x and y position based on its speed and angle using trigonometry.
  + Updates self.rect.x and self.rect.y to reflect the new position.
* **draw(self, screen)**:
  + Draws the bullet as a black circle on the screen.

**5. enemy.py**

This file defines the Enemy class, representing the hostile creatures the beaver must shoot.

**Class: Enemy**

* **\_\_init\_\_(self, score=0)**:
  + Initializes the enemy's starting x position (off-screen right) and sets its width and height.
  + **Spawning Vertical Limits:** Calculates self.min\_y (upper bound) and self.max\_y (lower bound) for spawning, using constants.ENEMY\_MIN\_Y and constants.HEIGHT - self.height to ensure enemies spawn within the playable area and are fully visible.
  + Randomly sets the enemy's initial y position within these calculated bounds.
  + Loads and scales the enemy image (vidra.png).
  + Sets the enemy's speed based on the current score (passed from GameManager), implementing difficulty scaling using constants.STAGE\_ONE\_SPEED and constants.STAGE\_TWO\_SPEED.
  + Creates self.rect for collision detection.
* **update(self)**:
  + Decreases the enemy's x position by its speed, moving it left across the screen.
  + Updates self.rect.x.
  + **Vertical Movement Constraints:** Uses max() and min() with self.min\_y and self.max\_y to ensure the enemy stays within its defined vertical movement bounds, preventing it from going into the HUD or off the bottom of the screen during its movement.
  + Updates self.rect.y.
* **draw(self, screen)**:
  + Draws the enemy image on the screen.

**6. game\_manager.py**

This file defines the GameManager class, which is responsible for managing the overall game state, updating all game objects, handling collisions, spawning enemies, and rendering the Heads-Up Display (HUD) and game over screen.

**Class: GameManager**

* **\_\_init\_\_(self)**:
  + **Dynamic UI Boundary Calculation:** This is a crucial step. It calculates calculated\_min\_y\_for\_game\_area based on the height of the ammo text and a vertical offset. This value defines the top edge of the playable game area.
  + **Updates Global Constants:** It then assigns this calculated value to constants.ENEMY\_MIN\_Y and constants.BEAVER\_MAX\_Y\_UPPER. This ensures that the Beaver and Enemy classes, when initialized or updated, use the correct boundaries that respect the HUD.
  + Initializes self.beaver, self.bullets (list), self.enemies (list), self.score, self.game\_over flag, spawn\_counter, and spawn\_interval.
  + Sets up fonts for general text (self.font) and the reloading message (self.reloading\_font).
  + Loads the screaming\_beaver.mp3 sound and initializes self.scream\_played to ensure it plays only once on game over.
  + Loads and scales the endImg.jpeg image to be displayed on game over.
* **update(self)**:
  + **Game Over State:** If self.game\_over is True, it plays the scream sound (once), clears all enemies, and stops further game updates.
  + **Reloading Logic:** Manages the beaver's reloading process, checking if RELOAD\_TIME\_SECONDS has passed to replenish ammo.
  + Calls self.beaver.update() to update the player character's state.
  + **Bullet Management:** Iterates through self.bullets, updates each bullet's position, and removes bullets that go off-screen or into the HUD area.
  + **Enemy Spawning:** Increments self.spawn\_counter and spawns a new Enemy (passing the current self.score for difficulty scaling) when self.spawn\_interval is reached.
  + **Enemy Updates & Merging:** Iterates through self.enemies, calls enemy.update(), and includes logic to prevent enemies from overlapping by adjusting their x position if they collide with a preceding enemy.
  + **Enemy Off-Screen Penalty:** Removes enemies that go off-screen to the left and penalizes the player's score. If the score drops to 0, the game ends.
  + **Collision Detection (Enemy-Beaver):** Checks for collisions between enemies and the beaver. If a collision occurs, the beaver loses HP, the enemy is removed, and if HP drops to 0, self.game\_over is set to True.
  + **Collision Detection (Bullet-Enemy):** Checks for collisions between bullets and enemies. If a bullet hits an enemy, both are removed, and the player's self.score increases.
* **draw(self, screen)**:
  + **Game Over Screen:** If self.game\_over is True, it draws the end\_image as the background and then renders a large, outlined, and shadowed "Game Over! Press R to Restart" message in the center.
  + **Active Game Screen:** If the game is active, it fills the screen with white, then draws the beaver, all active bullets, and all active enemies.
  + **HUD Rendering:** Draws the HP, Ammo, Stage, and Score text on the screen.
  + Displays a "RELOADING..." message if the beaver is currently reloading.
* **handle\_events(self)**:
  + Processes Pygame events such as quitting the game (pygame.QUIT).
  + Handles keyboard input:
    - K\_SPACE: Triggers the beaver's shoot() method. If shooting is successful, the bullet is added to self.bullets. If shooting fails due to no ammo, it initiates a reload.
    - K\_r: Restarts the game if self.game\_over is True, or initiates a manual reload if the beaver is not already reloading and has less than max ammo.
  + Returns False if the game should quit, True otherwise.