Links

- Final version of program - requires constants.py and images folder to be downloaded
- constants.py
- Images directory
- Version control evidence

Relevant Implications

Functionality

Functionality involves ensuring that the outcome functions as intended. For this project, this means creating a playable Llama game which is easily playable with no bugs. This means that the program cannot crash on edge cases, must be able to handle all possible events, etc.

Functionality is important because user experience is the most important factor in a developed outcome, and if the outcome does not have basic functionality, users will become frustrated. Additionally, the program will have no real use, as it is not able to function for its intended task.

Useability

Useability involves making it easy for the user to use the program without requiring help or assistance from others. For this project, this means making it simple to play, with intuitive controls and instructions that explain how the game works. This means everything must be designed so someone who has never touched a computer before would be able to play the game with as little of a learning curve as possible.

Useability is important because it is another key factor in the user experience. Users must feel like they understand the game, otherwise they will get frustrated and stop playing. This will result in the program not being used.

Aesthetics

Aesthetics involves making the game appealing to look at. For this project, this involves using images for various parts of the game, such as the background, obstacles, and character.

Aesthetics is important because users appreciate it when games have appealing graphics, and it makes the game look much more professional and developed.

Social

Social implications involve how the outcome affects users, and the wider community as a whole. Games which can be addictive need to have safeguards in place to stop users from spending too much time on the game. Another example is gambling games, which need safeguards to restrict how much the users can lose.

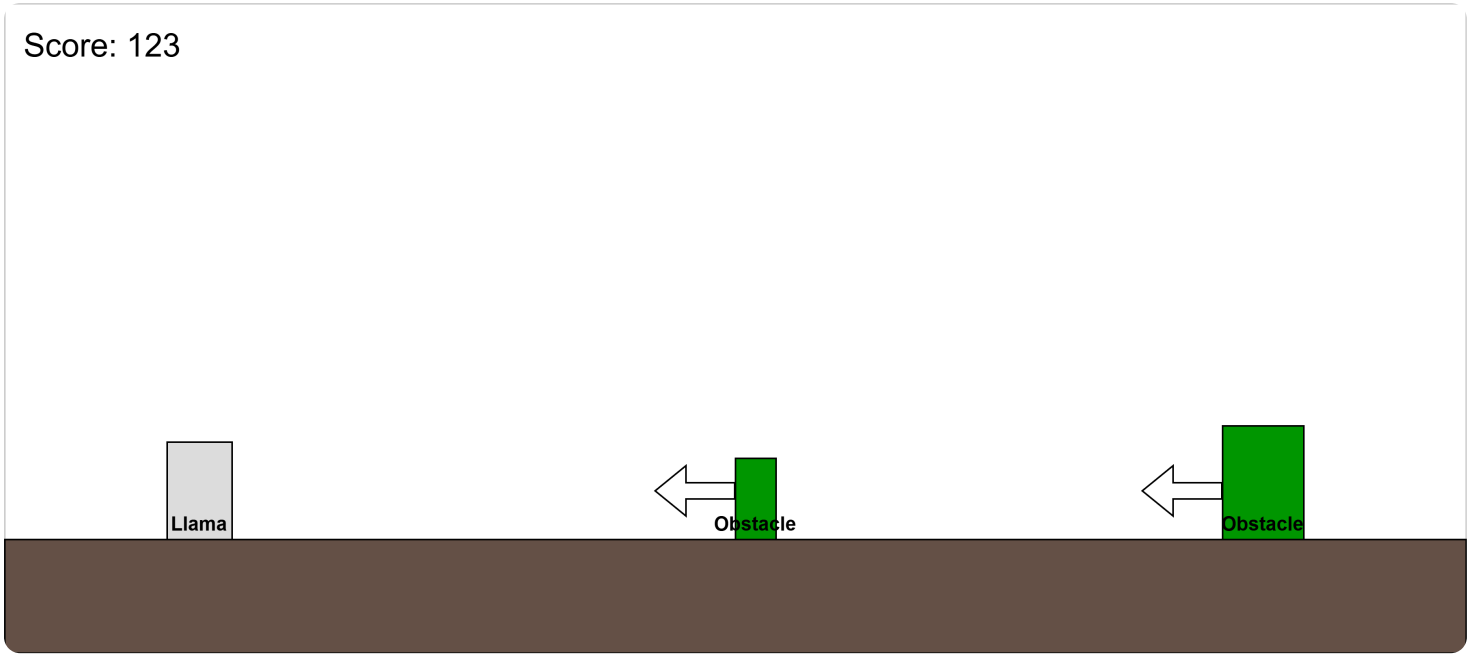
Social implications are important because games can have unintended negative consequences on their users.

Fortunately, the program which I will be creating is a very simple game, which means that additional safeguards will probably not need to be implemented to stop misuse of the game.

GUI Design

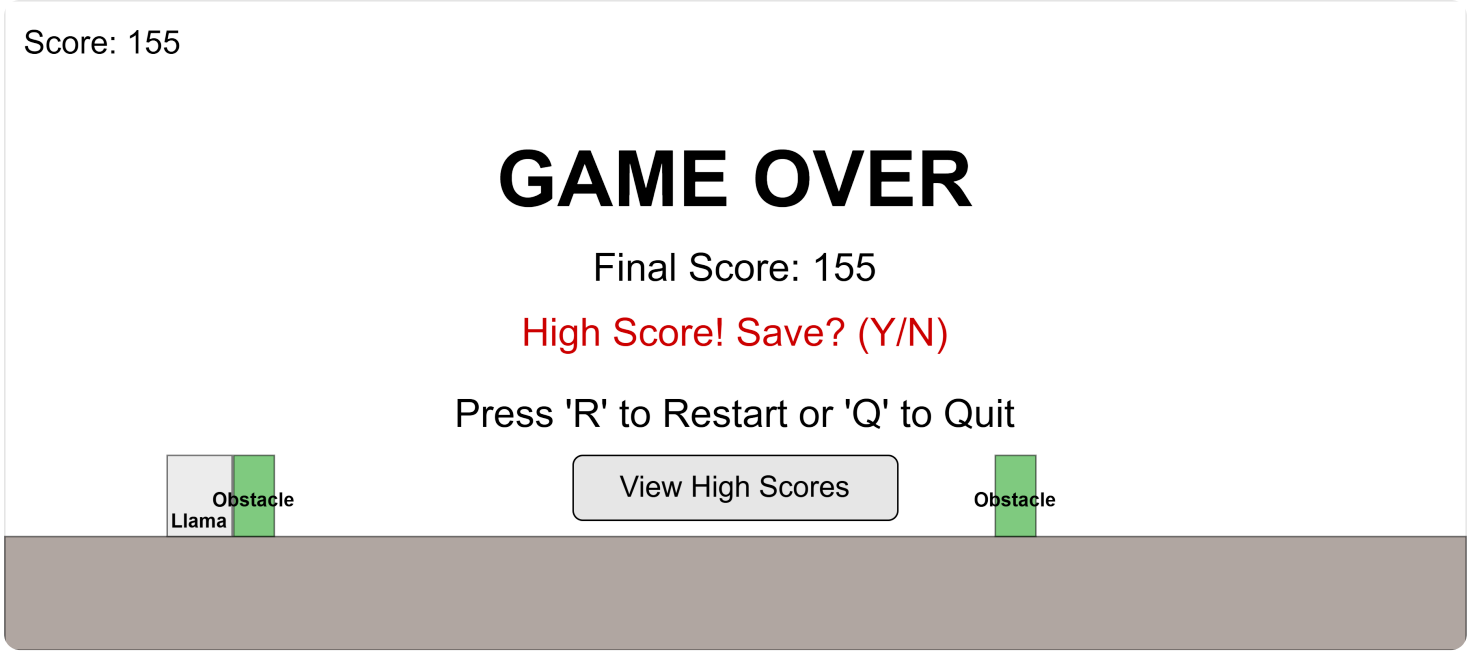
Wireframes

Regular Gameplay



Game over (not high score)

Game over (high score)



Enter name (high score)

High scores

High Scores

1. PLAYER - 210
2. PLAYER - 195
3. PLAYER - 180
4. PLAYER - 155
5. PLAYER - 140
6. PLAYER - 210
7. PLAYER - 195
8. PLAYER - 180
9. PLAYER - 155
10. PLAYER - 140

Press ESC to Return

Program Structure

Game Class

- Setup Game (`__init__`)
- Run Game Loop (`run`)
- Handle Events (`_handle_events`)
- Update Game State (`_update`)
- Draw Frame (`_draw`)
- Spawn Obstacle (`_spawn_obstacle`)
- Check Collisions (`_check_collisions`)
- Reset Game (`_reset_game`)
- Load High Scores (`_load_high_scores`)
- Save High Scores (`_save_high_scores`)
- Check Score Eligibility (`_check_score_eligible`)
- Add High Score (`_add_high_score`)
- Draw High Scores Screen (`_draw_high_scores_screen`)

Llama Class

- Setup Player (`__init__`)
- Update Player State (`update`)
- Perform Jump (`jump`)
- Reset Player (`reset`)

Obstacle Class

- Setup Obstacle (`__init__`)
- Update Obstacle State (`update`)

Scoreboard Class

- Setup Scoreboard (`__init__`)
- Update Score (`update`)
- Draw Score (`draw`)
- Reset Score (`reset`)

Project Decomposition

constants.py - Llama Game Decomposition

Define Constants

- [ ] Set the width and height of the game window.

- [ ] Set the target frames per second for the game.
- [ ] Define the vertical position of the ground.
- [ ] Define a unique signal (custom event) for when to create a new obstacle.
- [ ] Define the strength of gravity affecting the player.
- [ ] Define how high the player jumps.
- [ ] Set the starting horizontal position for the player.
- [ ] Set the initial speed for obstacles moving left.
- [ ] Set how often new obstacles appear (in milliseconds).
- [ ] Define standard colour values (like white, black, grey).
- [ ] (If using images) Specify the file locations for image assets.

Game Class - Llama Game Decomposition

Setup Game (`__init__`)

- [ ] Start up Pygame systems.
- [ ] Prepare the sound system (Optional: if using sound).
- [ ] Prepare the font system for displaying text.
- [ ] Create the main game window.
- [ ] Set the title displayed on the game window.
- [ ] Create a timer to control game speed (FPS).
- [ ] Set initial game status flags (e.g., running, game over, entering name, showing scores).
- [ ] Record the time the game session started for scoring.
- [ ] Create containers (groups) to hold all game objects and obstacles.
- [ ] Create the player character (Llama).
- [ ] Add the player character to the container for all game objects.
- [ ] Create the scoreboard display.
- [ ] Load or prepare background and ground visuals.
- [ ] Define a unique signal (custom event) for when to create a new obstacle.
- [ ] Start a timer that sends the obstacle creation signal repeatedly.
- [ ] Load high scores from the storage file (handle file not found).
- [ ] Prepare variables for player name input.
- [ ] Define the area/text for the "View High Scores" button.

Run Game Loop (`run`)

- [ ] Begin the main loop that keeps the game running.
- [ ] Handle player input and game events within the loop based on current game state.
- [ ] Update the state and position of all game objects within the loop (if applicable to state).
- [ ] Draw everything onto the screen within the loop based on current game state.
- [ ] Control the game's speed (frames per second) within the loop.

- [ ] Check if the game has ended after updating/drawing (sets game over state).
- [ ] Clean up Pygame resources after the main loop finishes.

Handle Events (`_handle_events`)

- [ ] Check for any user actions or game events that occurred.
- [ ] Check if the user tried to close the game window (always active).
- [ ] **If game is playing:**
  - [ ] Check if it's time to create a new obstacle.
  - [ ] If it's time, trigger the obstacle creation process.
  - [ ] Check if the user pressed the jump key; if so, make the player jump.
- [ ] **If game is over (and not entering name or showing scores):**
  - [ ] Check if score is eligible for saving; if so, listen for save confirmation (Y/N).
  - [ ] If 'Y' pressed, switch state to 'entering name'.
  - [ ] Check if the user pressed the restart key; if so, restart the game.
  - [ ] Check if the user pressed the quit key; if so, exit the game loop.
  - [ ] Check if the user clicked the "View High Scores" button; if so, switch state to 'showing scores'.
- [ ] **If entering name:**
  - [ ] Listen for letter/number key presses and update the temporary player name.
  - [ ] Listen for Backspace key to delete characters from the name.
  - [ ] Listen for Enter key to finalize name, save the score, and switch back to 'game over' state.
- [ ] **If showing scores:**
  - [ ] Listen for Escape key press to switch back to 'game over' state.

Update Game State (`_update`)

- [ ] Check if the game is in the 'playing' state.
- [ ] If playing, tell all game objects to update themselves.
- [ ] If playing, check if the player has collided with any obstacles (sets game over state).
- [ ] If playing, update the score based on elapsed time.
- [ ] (Optional) Add logic to make the game harder over time.

Draw Frame (`_draw`)

- [ ] Clear the screen or draw the main background.
- [ ] **If game is playing:**
  - [ ] Draw the ground element.
  - [ ] Draw all the active game objects (player, obstacles).
  - [ ] Draw the current score.

- [ ] **If game is over (and not entering name or showing scores):**
  - [ ] Draw the last frame of gameplay (ground, player, obstacles).
  - [ ] Draw the "Game Over" text.
  - [ ] Draw the final score text.
  - [ ] If score is eligible, draw the "Save Score? (Y/N)" prompt.
  - [ ] Draw the "View High Scores" button.
  - [ ] Draw the "Restart/Quit" instructions.
- [ ] **If entering name:**
  - [ ] Draw the last frame of gameplay (ground, player, obstacles).
  - [ ] Draw the "Enter Name: " prompt.
  - [ ] Draw the player name text as it's being typed.
- [ ] **If showing scores:**
  - [ ] Call the specific function to draw the high scores list.
- [ ] Show the final image on the display.

Spawn Obstacle (`_spawn_obstacle`)

- [ ] Create a new obstacle object.
- [ ] Add the new obstacle to the group of all active game objects.
- [ ] Add the new obstacle specifically to the group of obstacles.

Check Collisions (`_check_collisions`)

- [ ] Check if the player object is touching any obstacle object.
- [ ] Use precise collision detection.
- [ ] If a collision happened, set the game state to 'game over'.
- [ ] (Optional) If a collision happened, play a sound effect.

Reset Game (`_reset_game`)

- [ ] Set the game state back to 'playing' (or appropriate initial state).
- [ ] Reset the start time for the new game session.
- [ ] Reset the scoreboard to zero.
- [ ] Remove all obstacles currently on the screen.
- [ ] Put the player back in the starting position with reset physics.
- [ ] Reset name input variables.
- [ ] (Optional) Reset any difficulty scaling back to default.

Load High Scores (`_load_high_scores`)

- [ ] Define the filename for high scores storage.

- [ ] Try to open and read the high scores file using JSON.
- [ ] If successful, store the loaded list (ensure sorted).
- [ ] If file not found, create an empty list for high scores.
- [ ] Handle potential errors during file reading or JSON parsing.

Save High Scores (`_save_high_scores`)

- [ ] Define the filename for high scores storage.
- [ ] Try to open the high scores file for writing.
- [ ] Convert the current high scores list to JSON format and write it to the file.
- [ ] Handle potential errors during file writing.

Check Score Eligibility (`_check_score_eligible`)

- [ ] Get the player's final score.
- [ ] Compare the score against the loaded high scores list.
- [ ] Return true if the list has fewer than 10 scores OR if the player's score is higher than the 10th score. Otherwise, return false.

Add High Score (`_add_high_score`)

- [ ] Take the player's name and final score as input.
- [ ] Create a new score entry (dictionary or object).
- [ ] Add the new entry to the main high scores list.
- [ ] Sort the list by score (highest first).
- [ ] Trim the list to keep only the top 10 entries.
- [ ] Call the function to save the updated high scores list to the file.

Draw High Scores Screen (`_draw_high_scores_screen`)

- [ ] Clear the screen or draw a suitable background.
- [ ] Draw a title like "Top 10 High Scores".
- [ ] Loop through the loaded high scores list (up to 10).
- [ ] For each entry, format and draw the rank, name, and score.
- [ ] Draw instructions like "Press ESC to return".

Llama Class - Llama Game Decomposition

Setup Player (`__init__`)

- [ ] Initialize the base Sprite features.
- [ ] Load the player's visual appearance (image or shape).
- [ ] Set the initial visual appearance.
- [ ] Get the rectangle representing the player's position and size.
- [ ] Set the player's starting position on the screen.
- [ ] Remember the starting position for resetting later.
- [ ] Initialize physics variables (like vertical speed).
- [ ] Create a precise outline (mask) for collision detection.
- [ ] (Optional) Prepare frames for player animation.

Update Player State (`update`)

- [ ] Apply the effect of gravity to the player's vertical speed.
- [ ] Change the player's vertical position based on its current speed.
- [ ] Check if the player has landed on or fallen below the ground.
- [ ] If on the ground, stop downward movement and reset vertical speed.
- [ ] (Optional) Change the player's visual appearance based on state (jumping/running).
- [ ] (Optional) Update the collision mask if the visual appearance changed.

Perform Jump (`jump`)

- [ ] Check if the player is currently on the ground.
- [ ] If on the ground, give the player an upward vertical speed boost.
- [ ] (Optional) Change the player's visual appearance to jumping state.

Reset Player (`reset`)

- [ ] Move the player back to its initial starting position.
- [ ] Reset the player's vertical speed to zero.
- [ ] (Optional) Reset the player's visual appearance to the default (running) state.

Obstacle Class - Llama Game Decomposition

Setup Obstacle (`__init__`)

- [ ] Initialize the base Sprite features.
- [ ] Load the obstacle's visual appearance (image or shape).
- [ ] (Optional) Randomly choose which type of obstacle appearance to use.
- [ ] Set the obstacle's visual appearance.
- [ ] Get the rectangle representing the obstacle's position and size.

- [ ] Set the obstacle's starting position (off-screen to the right).
- [ ] Store the speed at which the obstacle should move.
- [ ] Create a precise outline (mask) for collision detection.

Update Obstacle State (`update`)

- [ ] Move the obstacle horizontally to the left based on its speed.
- [ ] Check if the obstacle has moved completely off the left side of the screen.
- [ ] If off-screen, remove the obstacle from the game.

Scoreboard Class - Llama Game Decomposition

Setup Scoreboard (`__init__`)

- [ ] Store the desired position and color for the score display.
- [ ] Load or prepare the font for rendering text.
- [ ] Set the initial score value to zero.
- [ ] Prepare variables to hold the rendered score text image and its position.
- [ ] Create the initial score text image (e.g., "Score: 0").

Update Score (`update`)

- [ ] Calculate the current score based on how long the game has been running.
- [ ] Check if the calculated score is different from the currently displayed score.
- [ ] If the score has changed, store the new score value.
- [ ] If the score has changed, create a new text image for the updated score.
- [ ] If the score has changed, update the position rectangle for the new text image.

Draw Score (`draw`)

- [ ] Draw the current score text image onto the main game screen.

Reset Score (`reset`)

- [ ] Set the score value back to zero.
- [ ] Create the text image for the zero score.
- [ ] Update the position rectangle for the zero score text.

Project Development

constants.py

Component Planning

Define Constants

- [ ] Set the width and height of the game window.
- [ ] Set the target frames per second for the game.
- [ ] Define the vertical position of the ground.
- [ ] Define a unique signal (custom event) for when to create a new obstacle.
- [ ] Define the strength of gravity affecting the player.
- [ ] Define how high the player jumps.
- [ ] Set the starting horizontal position for the player.
- [ ] Set the initial speed for obstacles moving left.
- [ ] Set how often new obstacles appear (in milliseconds).
- [ ] Define standard colour values (like white, black, grey).
- [ ] (If using images) Specify the file locations for image assets.

Change List

Date	Change
28/04	Added constant for the window title/caption `WINDOW_TITLE = "Llama Game - Joseph Surrey"`
1/05	Moved obstacle creation signal from `Game.__init__` to `constants.py`

Test Plan: constants.py

Test Results

Test 01 - 26/04

Test Results - constants.py - test_01.html

The program passed 14/18 tests successfully. The program failed 4/18 tests. The 4 tests that were failed were the image loading tests. The issue causing the failure was the image path in `constants.py`.

```
# Image file locations
PLAYER_IMAGE = "/images/Llama.png"
OBSTACLE_IMAGE = "/images/cactus.png"
GROUND_IMAGE = "/images/ground.png"
GAME_ICON = "/images/llama_icon.png"
```

The file locations linked to `/images`, which is an absolute path, looking for the image in the root directory of the hard drive. The image path should actually be

```
# Image file locations
PLAYER_IMAGE = "images/Llama.png"
OBSTACLE_IMAGE = "images/cactus.png"
GROUND_IMAGE = "images/ground.png"
GAME_ICON = "images/llama_icon.png"
```

This ensures that the program looks for the files in the `images` folder in the current working directory.

Test 02 - 26/04

Test Results - constants.py - test_02.html

The program passed 18/18 tests successfully after making the changes from Test 01.

Game Class - Llama Game Decomposition

Setup Game (`__init__`)

Component Planning

Setup Game (`__init__`)

- [ ] Start up Pygame systems.
- [ ] Prepare the sound system (Optional: if using sound).
- [ ] Prepare the font system for displaying text.
- [ ] Create the main game window.
- [ ] Set the title displayed on the game window.
- [ ] Create a timer to control game speed (FPS).
- [ ] Set initial game status flags (e.g., running, game over, entering name, showing scores).
- [ ] Record the time the game session started for scoring.
- [ ] Create containers (groups) to hold all game objects and obstacles.
- [ ] Create the player character (Llama).
- [ ] Add the player character to the container for all game objects.
- [ ] Create the scoreboard display.
- [ ] Load or prepare background and ground visuals.
- [ ] Define a unique signal (custom event) for when to create a new obstacle.
- [ ] Start a timer that sends the obstacle creation signal repeatedly.
- [ ] Load high scores from the storage file (handle file not found).
- [ ] Prepare variables for player name input.

- [ ] Define the area/text for the "View High Scores" button.

Change List

Date	Change
1/05	Moved obstacle creation signal from `Game.__init__` to `constants.py`

Test Plan

Test Case	Verification Focus	Expected Output	Test Typ
Standard Initialization (Pygame Modules)	`Game()` called	`pygame.init()`, `pygame.mixer.init()`, `pygame.font.init()` are called exactly once.	Mock Check
Standard Initialization (Screen & Caption)	`Game()` called	`pygame.display.set_mode` called with `(WINDOW_WIDTH, WINDOW_HEIGHT)`. `pygame.display.set_caption` called with `WINDOW_TITLE`. `self.screen` is assigned the result of `set_mode`.	Mock Check
Standard Initialization (Clock & Start Time)	`Game()` called	`self.clock` is assigned `pygame.time.Clock`. `self.start_time` is assigned the result of `pygame.time.get_ticks()`.	Attribute Check
Attribute Value Checks (Flags, Name)	`Game()` called	`self.running` is True, `self.game_over` is False, `self.entering_name` is False, `self.displaying_scores` is False, `self.score_eligible_for_save` is False, `self.player_name` is "".	Attribute Check
Component Instantiation (Llama, Scoreboard)	`Game()` called	`Llama` class is instantiated. `Scoreboard` class is instantiated. `self.llama` and `self.scoreboard` hold the respective instances.	Mock Check/A
Sprite Group Setup	`Game()` called	`self.all_sprites` and `self.obstacles` are instances of `pygame.sprite.Group`. Mock `Llama` instance is added to `self.all_sprites`.	Type/St Check
Ground Image Load Success	`pygame.image.load` succeeds	`pygame.image.load` called with `constants.GROUND_IMAGE`. `convert()` is called on the result. `self.ground_image` holds the result of `convert()`.	Mock/At Check
Ground Image Load Failure (pygame.error)	`pygame.image.load` raises `pygame.error`	Exception is caught, `self.ground_image` is set to `None`.	Error Handlin
Ground Image Load Failure (FileNotFoundError)	`pygame.image.load` raises `FileNotFoundError`	Exception is caught, `self.ground_image` is set to `None`.	Error Handlin

Test Case	Verification Focus	Expected Output	Test Typ
Obstacle Timer Set	`Game()` called	`pygame.time.set_timer` called once with `constants.OBSTACLE_SPAWN_EVENT` and `constants.OBSTACLE_CREATION_INTERVAL`.	Mock Check
High Score Load Called	`Game()` called	Instance's `_load_high_scores()` method is called exactly once.	Mock Check
Font Object Creation	`Game()` called	`pygame.font.SysFont` is called for each font (`score_font`, `game_over_font`, `button_font`, `input_font`) with `(None, expected_size)`. Corresponding attributes are set.	Mock Check

Test Results

Test 01

Test Results - game_init - test_01.html

Passed 12/12 tests

Run Game Loop (`run`)

Component Planning

Run Game Loop (`run`)

- [ ] Begin the main loop that keeps the game running.
- [ ] Handle player input and game events within the loop based on current game state.
- [ ] Update the state and position of all game objects within the loop (if applicable to state).
- [ ] Draw everything onto the screen within the loop based on current game state.
- [ ] Control the game's speed (frames per second) within the loop.
- [ ] Check if the game has ended after updating/drawing (sets game over state).
- [ ] Clean up Pygame resources after the main loop finishes.

Test Plan

Test Case	Input / Conditions	Expected Output	Test Type
Normal Loop Execution & Exit	`self.running` starts True, then becomes False in `_handle_events`	`_handle_events`, `_update`, `_draw`, `clock.tick_busy_loop` called at least once. Loop terminates. `pygame.quit()` and `sys.exit()` called after loop.	Mock Check

Test Case	Input / Conditions	Expected Output	Test Type
Clean Exit Sequence	Loop terminates (e.g., `self.running` becomes False)	`pygame.quit()` and `sys.exit()` are called *after* the loop finishes.	Mock Check
Clock Ticking Verified	Loop runs for at least one iteration	`game.clock.tick_busy_loop` is called with `constants.FPS`.	Mock Check
Exception within Loop	`_update` (or another sub-method) raises Exception	Loop terminates abruptly. `pygame.quit()` and `sys.exit()` are *not* called by the `run` method itself.	Error Case

Test Results

Test 01

Test Results - game_run - test_01.html

Passed 4/4 tests

Handle Events (`_handle_events`)

Component Planning

Handle Events (`_handle_events`)

- [ ] Check for any user actions or game events that occurred.
- [ ] Check if the user tried to close the game window (always active).
- [ ] **If game is playing:**
  - [ ] Check if it's time to create a new obstacle.
  - [ ] If it's time, trigger the obstacle creation process.
  - [ ] Check if the user pressed the jump key; if so, make the player jump.
- [ ] **If game is over (and not entering name or showing scores):**
  - [ ] Check if score is eligible for saving; if so, listen for save confirmation (Y/N).
  - [ ] If 'Y' pressed, switch state to 'entering name'.
  - [ ] Check if the user pressed the restart key; if so, restart the game.
  - [ ] Check if the user pressed the quit key; if so, exit the game loop.
  - [ ] Check if the user clicked the "View High Scores" button; if so, switch state to 'showing scores'.
- [ ] **If entering name:**
  - [ ] Listen for letter/number key presses and update the temporary player name.
  - [ ] Listen for Backspace key to delete characters from the name.
  - [ ] Listen for Enter key to finalize name, save the score, and switch back to 'game over' state.

- [ ] **If showing scores:**
  - [ ] Listen for Escape key press to switch back to 'game over' state.

Test Plan

Test Case	Input / Conditions	Expected Output	Test Type
Quit Event (Window Close)	User clicks window 'X' button	`pygame.QUIT` event detected, `self.running` set to False.	Expected
Jump Key Press (Playing - Space)	SPACE pressed while `game_over` is False	`llama.jump()` method is called.	Expected
Jump Key Press (Playing - Up)	UP pressed while `game_over` is False	`llama.jump()` method is called.	Expected
Jump Key Press (Game Over)	SPACE or UP pressed while `game_over` is True	No action (jump not called).	Expected
Restart Key Press (Game Over)	'R' pressed while `game_over` is True	`_reset_game()` method is called.	Expected
Restart Key Press (Playing)	'R' pressed while `game_over` is False	No action.	Expected
Quit Key Press (Game Over)	'Q' pressed while `game_over` is True	`self.running` set to False.	Expected
Quit Key Press (Playing)	'Q' pressed while `game_over` is False	No action.	Expected
Obstacle Spawn Event (Playing)	`pygame.USEREVENT + 1` occurs while `game_over` is False	`_spawn_obstacle()` method is called.	Expected
Obstacle Spawn Event (Game Over)	`pygame.USEREVENT + 1` occurs while `game_over` is True	No action (`_spawn_obstacle` not called).	Expected
Save Score Confirm (Y - Eligible)	'Y' pressed while `game_over` is True & score is eligible	Game state changes to `entering_name`, `player_name` reset to "".	Expected
Save Score Decline (N - Eligible)	'N' pressed while `game_over` is True & score is eligible	`score_eligible_for_save` set to False.	Expected
Save Score Keys (Not Eligible)	'Y' or 'N' pressed while `game_over` is True & score not eligible	No state change (`entering_name` remains False, `score_eligible_for_save` remains False).	Expected
Enter Name Keys (Alphanumeric)	Alphanumeric keys pressed while `entering_name`	Characters appended to `self.player_name`.	Expected
Enter Name Keys (Space)	Space key pressed while `entering_name`	Space appended to `self.player_name`.	Expected

Test Case	Input / Conditions	Expected Output	Test Type
Enter Name - Length Limit	Key pressed when `len(self.player_name)` is 6	Character not appended to `self.player_name`.	Boundary
Backspace Key (Entering Name State)	BACKSPACE pressed while `entering_name`	Last character removed from `self.player_name`.	Expected
Enter Key (Entering Name State - Valid)	ENTER pressed while `entering_name` with non-empty `player_name`	`_add_high_score()` called with name/score, `entering_name` set to False, `score_eligible_for_save` set to False.	Expected
Enter Key (Entering Name State - Empty)	ENTER pressed while `entering_name` with empty `player_name`	`_add_high_score()` not called, `entering_name` set to False, `score_eligible_for_save` set to False.	Boundary
Escape Key (Showing Scores State)	ESC pressed while `displaying_scores`	`displaying_scores` set to False.	Expected
Unexpected Key Press (Playing)	Any key other than SPACE/UP pressed while playing	No action.	Unexpected Input
Unexpected Key Press (Game Over - Standard)	Any key other than R, Q, Y, N pressed while game over	No action.	Unexpected Input
Unexpected Key Press (Entering Name)	Any key other than alphanum, space, backspace, enter pressed	No action / character not added.	Unexpected Input
Unexpected Key Press (Showing Scores)	Any key other than ESC pressed while showing scores	No action.	Unexpected Input
Multiple Events Per Frame	e.g., Key press and Obstacle Spawn event in same frame	Both events processed correctly in sequence within the loop iteration.	Edge Case

Test Results - game__handle_events - test_01.html

Test Results

Test Results - game__handle_events - test_01.html

Passed 25/25 tests

Update Game State (`_update`)

Component Planning

Update Game State (`_update`)

- [ ] Check if the game is in the 'playing' state.
- [ ] If playing, tell all game objects to update themselves.
- [ ] If playing, check if the player has collided with any obstacles (sets game over state).
- [ ] If playing, update the score based on elapsed time.
- [ ] (Optional) Add logic to make the game harder over time.

Test Plan

Test Case	Input / Conditions	Expected Output	Test Type
Update While Playing	`game_over`, `entering_name`, `displaying_scores` are all False	`all_sprites.update()`, `scoreboard.update()`, `_check_collisions()` are called.	Mock Check
Update While Game Over	`game_over` is True	`all_sprites.update()`, `scoreboard.update()`, `_check_collisions()` are *not* called.	Mock Check
Update While Entering Name	`entering_name` is True	`all_sprites.update()`, `scoreboard.update()`, `_check_collisions()` are *not* called.	Mock Check
Update While Displaying Scores	`displaying_scores` is True	`all_sprites.update()`, `scoreboard.update()`, `_check_collisions()` are *not* called.	Mock Check
No Sprites Present	`all_sprites` group is empty, game is playing	`all_sprites.update()` is called without error. `scoreboard.update()` and `_check_collisions()` are also called.	Edge/Mock Check
Verify `scoreboard.update` Arguments	`game_over`, `entering_name`, `displaying_scores` are all False	`scoreboard.update()` called with `pygame.time.get_ticks()` result and `game.start_time`.	Mock Check

Test Results

Test 01

Test Results - game__update - test_01.html

Passed 6/6

Draw Frame (`_draw`)

Component Planning

Draw Frame (`_draw`)

- [ ] Clear the screen or draw the main background.
- [ ] **If game is playing:**
  - [ ] Draw the ground element.
  - [ ] Draw all the active game objects (player, obstacles).
  - [ ] Draw the current score.
- [ ] **If game is over (and not entering name or showing scores):**
  - [ ] Draw the last frame of gameplay (ground, player, obstacles).
  - [ ] Draw the "Game Over" text.
  - [ ] Draw the final score text.
  - [ ] If score is eligible, draw the "Save Score? (Y/N)" prompt.
  - [ ] Draw the "View High Scores" button.
  - [ ] Draw the "Restart/Quit" instructions.
- [ ] **If entering name:**
  - [ ] Draw the last frame of gameplay (ground, player, obstacles).
  - [ ] Draw the "Enter Name: " prompt.
  - [ ] Draw the player name text as it's being typed.
- [ ] **If showing scores:**
  - [ ] Call the specific function to draw the high scores list.
- [ ] Show the final image on the display.

Test Plan

Test Case	Input / Conditions	Expected Output / Checks	Test Type
Draw While Playing (Ground Image)	`game_over`, `entering_name`, `displaying_scores` all False. `ground_image` exists.	`screen.fill`, `screen.blit` (ground), `all_sprites.draw`, `scoreboard.draw` called. `font.render` for Game Over text *not* called.	Expected
Draw While Playing (No Ground Image)	`game_over`, `entering_name`, `displaying_scores` all False. `ground_image` is None.	`screen.fill`, `pygame.draw.rect` (fallback ground), `all_sprites.draw`, `scoreboard.draw` called. `font.render` for Game Over text *not* called.	Edge/Error

Test Case	Input / Conditions	Expected Output / Checks	Test Type
Draw Game Over (Not Eligible)	`game_over` True, `entering_name` False, `displaying_scores` False, `score_eligible_for_save` False. `ground_image` exists.	`screen.fill`, `screen.blit` (ground), `all_sprites.draw`, `scoreboard.draw` called. Game Over, Final Score, Instructions text rendered and blitted. "Save Score?" prompt *not* rendered.	Expected
Draw Game Over (Eligible)	`game_over` True, `entering_name` False, `displaying_scores` False, `score_eligible_for_save` True. `ground_image` exists.	`screen.fill`, `screen.blit` (ground), `all_sprites.draw`, `scoreboard.draw` called. Game Over, Final Score, "Save Score? (Y/N)", Instructions text rendered and blitted.	Expected
Draw Game Over (No Ground Image)	`game_over` True, `entering_name` False, `displaying_scores` False, `score_eligible_for_save` False. `ground_image` is None.	`screen.fill`, `pygame.draw.rect` (fallback ground), `all_sprites.draw`, `scoreboard.draw` called. Game Over, Final Score, Instructions text rendered and blitted.	Edge/Error
State: Entering Name	`entering_name` True. Others False.	`screen.fill` called. Gameplay/Game Over drawing block skipped (`screen.blit` for ground, `all_sprites.draw`, `scoreboard.draw`, `font.render` for Game Over not called).	State Check

Test Results

Test 01

Test Results - game__draw - test_01.html

Passed 9/9 tests

Spawn Obstacle (`_spawn_obstacle`)

Component Planning

Spawn Obstacle (`_spawn_obstacle`)

- [ ] Create a new obstacle object.
- [ ] Add the new obstacle to the group of all active game objects.
- [ ] Add the new obstacle specifically to the group of obstacles.

Test Plan

Spawn Obstacle (`_spawn_obstacle`)

Component Planning

Spawn Obstacle (`_spawn_obstacle`)

- [ ] Create a new obstacle object.
- [ ] Add the new obstacle to the group of all active game objects.
- [ ] Add the new obstacle specifically to the group of obstacles.

Test Plan

Test Case	Input / Conditions	Expected Output	Test Type
Standard Spawn	Called from event handler	New `Obstacle` instance created. Instance added to `self.all_sprites` and `self.obstacles` groups.	Expected

Test Results

Test Results - game__spawn_obstacle.html

Passed 1/1 tests

Check Collisions (`_check_collisions`)

Component Planning

Check Collisions (`_check_collisions`)

- [ ] Check if the player object is touching any obstacle object.
- [ ] Use precise collision detection.
- [ ] If a collision happened, set the game state to 'game over'.
- [ ] (Optional) If a collision happened, play a sound effect.

Test Plan

Test Case	Input / Conditions	Expected Output / Checks	Test Type
No Collision	`pygame.sprite.spritecollide` returns empty list.	`game_over` remains `False`. `_check_score_eligible` is *not*	Expected

Test Case	Input / Conditions	Expected Output / Checks	Test Type
		called. `pygame.time.set_timer` is *not* called with `(OBSTACLE_SPAWN_EVENT, 0)`.	
Collision Occurs (Score Eligible)	`spritecollide` returns non-empty list. Mock `_check_score_eligible` returns `True`.	`game_over` becomes `True`. `_check_score_eligible` is called once. `score_eligible_for_save` becomes `True`. `pygame.time.set_timer` is called once with `(OBSTACLE_SPAWN_EVENT, 0)`.	Expected/Mock
Collision Occurs (Score Not Eligible)	`spritecollide` returns non-empty list. Mock `_check_score_eligible` returns `False`.	`game_over` becomes `True`. `_check_score_eligible` is called once. `score_eligible_for_save` becomes `False`. `pygame.time.set_timer` is called once with `(OBSTACLE_SPAWN_EVENT, 0)`.	Expected/Mock
No Obstacles Present	`self.obstacles` group is empty.	`pygame.sprite.spritecollide` returns empty list. `game_over` remains `False`. `_check_score_eligible` is *not* called. `pygame.time.set_timer` is *not* called with `(OBSTACLE_SPAWN_EVENT, 0)`. No error occurs.	Edge Case/Mock

Test Results

Test 01

Test Results - game__check_collisions - test_01.html

Passed 4/4 tests

Reset Game (`_reset_game`)

Component Planning

Reset Game (`_reset_game`)

- [ ] Set the game state back to 'playing' (or appropriate initial state).
- [ ] Reset the start time for the new game session.

- [ ] Reset the scoreboard to zero.
- [ ] Remove all obstacles currently on the screen.
- [ ] Put the player back in the starting position with reset physics.
- [ ] Reset name input variables.
- [ ] (Optional) Reset any difficulty scaling back to default.

Test Plan

Test Case	Input / Conditions	Expected Output / Checks	Test Type
Standard Reset	Called, e.g., from Game Over state	`game_over`, `entering_name`, `displaying_scores`, `score_eligible_for_save` are False. `start_time_ticks` is updated (mock `pygame.time.get_ticks`). `scoreboard.reset()` called. `llama.reset()` called. `player_name` is "".	State/Mock Check
Obstacle Group Reset	`obstacles` group has sprites before call	`obstacles.empty()` is called. The `obstacles` group is empty after the call.	State/Mock Check
All Sprites Group Reset	`all_sprites` has llama and obstacles before call	`all_sprites.empty()` is called. `all_sprites.add(llama)` is called. After reset, `all_sprites` group contains *only* the `llama` instance.	State/Mock Check
Obstacle Timer Restart	Called	`pygame.time.set_timer` is called once with `constants.OBSTACLE_SPAWN_EVENT` and `constants.OBSTACLE_CREATION_INTERVAL`.	Mock Check
Call Reset While Playing	Called when `game_over` is False	Method executes without crashing. All state variables (`game_over`, `entering_name`, etc.) are reset as per "Standard Reset".	Robustness/State
Reset with Empty Obstacles	`obstacles` group is already empty before call	`obstacles.empty()` is called without error. Group remains empty. Other reset actions occur normally.	Edge Case/Mock Check

Test Results

Test 01

Test Results - game__reset_game - test_01.html

Passed 7/7 tests

Load High Scores (`_load_high_scores`)

Component Planning

Load High Scores (`_load_high_scores`)

- [ ] Define the filename for high scores storage.
- [ ] Try to open and read the high scores file using JSON.
- [ ] If successful, store the loaded list (ensure sorted).
- [ ] If file not found, create an empty list for high scores.
- [ ] Handle potential errors during file reading or JSON parsing.

Test Plan

Test Case	Input / Conditions	Expected Output / Checks	Test Type
File Exists, Valid JSON List	`high_scores.json` contains `[{"name":"A","score":10}]`	Returns the list `[{'name':'A', 'score':10}]`.	Expected
File Exists, Correct Sorting	`high_scores.json` contains `[{"name":"B","score":5}, {"name":"A","score":10}]`	Returns the sorted list `[{'name':'A', 'score':10}, {'name':'B', 'score':5}]`.	Expected
File Exists, Missing Keys	`high_scores.json` contains `[{"name":"A"}, {"name":"B", "score":10}]`	Returns sorted list `[{'name':'B', 'score':10}, {'name':'A', 'score':0}]` (uses default 0 for missing score). No error.	Expected
File Exists, Truncation	`high_scores.json` contains 12 valid score entries	Returns only the top 10 highest score entries, correctly sorted.	Boundary
File Exists, Empty JSON List	`high_scores.json` contains `[]`	Returns `[]`.	Expected
File Does Not Exist	No `high_scores.json` file	Returns `[]`. No error. `is_file()` check prevents attempt to open.	Edge Case
Path Is Directory	`high_scores.json` exists but is a directory	Returns `[]`. No error. `is_file()` check prevents attempt to open.	Edge Case
File Exists, Invalid JSON	`high_scores.json` contains `"abc"`	`json.JSONDecodeError` caught, returns `[]`. No crash. Print message potentially called.	Error Handling
File Exists, Content Not List	`high_scores.json` contains `{"name":"A","score":10}` (a dictionary)	`AttributeError` on sort caught by generic `except`. **Raises** `TypeError` when slicing the non-list `scores` variable in `return`.	Error Handling

Test Case	Input / Conditions	Expected Output / Checks	Test Type
File Exists, List w/ Non-Dicts	`high_scores.json` contains `[1, 2, 3]`	`AttributeError` on `item.get` caught by generic `except`. **Raises** `TypeError` when slicing the list `scores` variable in `return`.	Error Handling
File Exists, Permission Error	`high_scores.json` exists, but `open()` raises `OSError`	`OSError` caught by generic `except Exception`. Returns `[]`. Print message potentially called.	Error Handling

Test Results

Test 01

Test Results - game__load_high_scores.html

Passed 11/11 tests

Save High Scores (`_save_high_scores`)

Component Planning

Save High Scores (`_save_high_scores`)

- [ ] Define the filename for high scores storage.
- [ ] Try to open the high scores file for writing.
- [ ] Convert the current high scores list to JSON format and write it to the file.
- [ ] Handle potential errors during file writing.

Test Plan

Test Case	Input / Conditions	Expected Output	Test Type
Save Valid List	`self.high_scores` is `[{"name":"A","score":10}]`	`high_scores.json` created/overwritten with the correct JSON representation of the list. No errors raised.	Expected
Save Empty List	`self.high_scores` is `[]`	`high_scores.json` created/overwritten with `[]`. No errors raised.	Expected
Invalid Data in List	`self.high_scores` contains non-JSON serializable data	`TypeError` during `json.dump`, caught, save fails gracefully. Print message potentially called. No crash.	Error Handling

Test Case	Input / Conditions	Expected Output	Test Type
File Write IO Error	`open()` raises `IOError` when opening file for writing	`IOError` caught, save fails gracefully. Print message potentially called. No crash.	Error Handling

Test Results

Test 01

Test Results - game__save_high_scores - test_01.html

Passed 4/4 tests

Check Score Eligibility (`_check_score_eligible`)

Component Planning

Check Score Eligibility (`_check_score_eligible`)

- [ ] Get the player's final score.
- [ ] Compare the score against the loaded high scores list.
- [ ] Return true if the list has fewer than 10 scores OR if the player's score is higher than the 10th score. Otherwise, return false.

Test Plan

Test Case	Input / Conditions	Expected Output	Test Type
List < 10, Any Score	`len(self.high_scores)` is 5, `final_score` is 1	True	Expected
List = 10, Score > 10th	`len` is 10, 10th score is 50, `final_score` is 51	True	Expected
List = 10, Score = 10th	`len` is 10, 10th score is 50, `final_score` is 50	False	Boundary
List = 10, Score < 10th	`len` is 10, 10th score is 50, `final_score` is 49	False	Expected
Empty List	`self.high_scores` is `[]`, `final_score` is 10	True	Edge Case
List = 10, Last Entry Missing Score	`len` is 10, 10th entry is `{'name':'Bad'}`, `final_score` is 1	True	Robustness

Test Results

Test 01

Test Results - game__check_score_eligibility.html

Passed 7/7 tests

Add High Score (`_add_high_score`)

Component Planning

Add High Score (`_add_high_score`)

- [ ] Take the player's name and final score as input.
- [ ] Create a new score entry (dictionary or object).
- [ ] Add the new entry to the main high scores list.
- [ ] Sort the list by score (highest first).
- [ ] Trim the list to keep only the top 10 entries.
- [ ] Call the function to save the updated high scores list to the file.

Test Plan

Add High Score (`_add_high_score`) - Updated

Test Case	Input / Conditions	Expected Output	Test Type
Add to Empty List	`name` ="A", `score` =10, `high_scores` = `[]`	`self.high_scores` becomes `[{'name':'A', 'score':10}]`. `_save_high_scores()` called.	Expected
Add to Short List	`name` ="B", `score` =5, `high_scores` = `[{'name':'A', 'score':10}]`	`self.high_scores` becomes `[{'name':'A', 'score':10}, {'name':'B', 'score':5}]` (sorted). `_save_high_scores()` called.	Expected
Add Higher Score	`name` ="C", `score` =15, `high_scores` = `[{'name':'A', 'score':10}]`	`self.high_scores` becomes `[{'name':'C', 'score':15}, {'name':'A', 'score':10}]` (sorted). `_save_high_scores()` called.	Expected
Add to Full List (Beats 10th)	`name` ="New", `score` =55, `high_scores` has 10 items, 10th score is 50	New score added, list sorted, list truncated back to 10 items (lowest score dropped). `_save_high_scores()` called.	Expected
Add to Full List (Not Top 10)	`name` ="Low", `score` =45, `high_scores` has 10	(Defensive) Score added, list sorted, original 10th score	Defensive Test

Test Case	Input / Conditions	Expected Output	Test Type
	items, 10th score is 50	dropped, list truncated to 10. `_save_high_scores()` called.	
Add Duplicate Score	`name` ="D", `score` =10, `high_scores` = `[{'name':'A', 'score':10}]`	New score added, list sorted (order of duplicates may vary based on sort stability), list length increases. `_save_high_scores()` called.	Edge Case
Add with Name Stripping	`name` =" E ", `score` =20, `high_scores` = `[]`	`self.high_scores` becomes `[{'name':'E', 'score':20}]`. `_save_high_scores()` called.	**Added**
Add Making List Size Exactly 10	`name` ="Tenth", `score` =5, `high_scores` has 9 items (highest 100, lowest 10)	New score added as 10th item, list sorted, list length becomes 10. `_save_high_scores()` called.	**Added Boundary**
Add Non-Numeric Score	`name` ="F", `score` ="abc", `high_scores` = `[]`	Score added `{'name':'F', 'score':'abc'}`, list sorted (likely placing 'abc' based on type comparison rules, potentially at the end if compared with ints). `_save_high_scores` called.	**Added Robustness**
Verify `_save_high_scores` Call	Any valid addition scenario	Mocked `_save_high_scores` method is called exactly once.	**Added Mock**
Invalid Name Input (None)	`name` =None, `score` =10	`AttributeError` occurs on `name.strip()`. Exception should be caught if robustness desired, otherwise test expects failure. (Current code will raise AttributeError).	Error Handling
Add Non-Numeric Score	`name` ="F", `score` ="abc", `high_scores` = `[{'name':'Num', 'score':50}]`	`TypeError` raised during sort due to comparison between `int` and `str`. `_save_high_scores` not called.	**Updated Error Handling**

Test Results

Test 01

Test Results - game__add_high_score.html

Passed 10/10 tests

Draw High Scores Screen (`_draw_high_scores_screen`)

Component Planning

Draw High Scores Screen (`_draw_high_scores_screen`)

- [ ] Clear the screen or draw a suitable background.
- [ ] Draw a title like "Top 10 High Scores".
- [ ] Loop through the loaded high scores list (up to 10).
- [ ] For each entry, format and draw the rank, name, and score.
- [ ] Draw instructions like "Press ESC to return".

Test Plan

Test Case	Input / Conditions	Expected Output / Checks	Test Type
Draw Empty List	`self.high_scores` is `[]`	`screen.fill` called with `constants.GREY`. Title "High Scores" rendered & blitted (correct font, color, position). "No high scores yet!" rendered & blitted (correct font, color, position). Return instruction rendered & blitted.	Edge Case
Draw Partial List (<10)	`self.high_scores` has 3 entries	`screen.fill` called with `constants.GREY`. Title rendered & blitted. 3 score entries rendered & blitted (correct rank, name, score format, font, color, positions). Return instruction rendered & blitted.	Expected
Draw Full List (10)	`self.high_scores` has 10 entries	`screen.fill` called with `constants.GREY`. Title rendered & blitted. 10 score entries rendered & blitted (correct rank, name, score format, font, color, positions). Return instruction rendered & blitted.	Expected
Draw Long Names/Scores	Entries have very long names or large scores	`font.render` called with the full long name/score strings. Blitting occurs at calculated positions (visual overflow not checked). Other elements (fill, title, return) drawn correctly.	Boundary
Test Malformed Entry	`self.high_scores` contains `[{"name": "A", "score": 10}, {"score": 5}, {"name": "C"}]`	`screen.fill`, Title, Return instruction drawn correctly. Entries rendered as "1. A - 10", "2. N/A - 5", "3. C - 0" using `.get()` defaults (correct font, color, positions).	Robustness/Error

Test Case	Input / Conditions	Expected Output / Checks	Test Type
Verify Element Positions	Any list state (e.g., partial list)	Title, score entries, empty message (if applicable), and return instruction are blitted at positions calculated using `center=(constants.WINDOW_WIDTH // 2, Y)` where Y depends on the element and loop index.	Layout Check
Verify Font/Color Usage	Any list state (e.g., partial list)	Title uses `highscore_title_font`. Entries use `highscore_entry_font`. Empty message uses `instruction_font`. Return instruction uses `button_font`. All text rendered with `constants.BLACK`.	Style/State Check

Test Results

Test 01

Test Results - game__draw_high_scores_screen - test_01.html

Passed 6/6 tests

Llama Class - Llama Game Decomposition

Setup Player (`__init__`)

Component Planning

Setup Player (`__init__`)

- [ ] Initialize the base Sprite features.
- [ ] Load the player's visual appearance (image or shape).
- [ ] Set the initial visual appearance.
- [ ] Get the rectangle representing the player's position and size.
- [ ] Set the player's starting position on the screen.
- [ ] Remember the starting position for resetting later.
- [ ] Initialize physics variables (like vertical speed).
- [ ] Create a precise outline (mask) for collision detection.
- [ ] (Optional) Prepare frames for player animation.

Test Plan

Test Case	Input / Conditions	Expected Output	Test Ty
Superclass Initialization	`Llama()` called	`pygame.sprite.Sprite.__init__(self)` is called exactly once.	Mock Check
Image Load Success Path	`pygame.image.load` succeeds	`pygame.image.load` called with `constants.PLAYER_IMAGE`. `convert_alpha()` called on result. `self.image` assigned the converted surface.	Mock/A Check
Image Load Failure Path (Fallback)	`pygame.image.load` raises `Exception`	`pygame.Surface` created with `[40, 60]`. `fill()` called with `constants.RED`. `self.image` assigned the fallback surface.	Error Handlin
Rect and Mask Creation (Success/Failure)	`Llama()` called (image loaded or fallback)	`self.image.get_rect()` called. `self.rect` assigned the result. `pygame.mask.from_surface` called with `self.image`. `self.mask` assigned the result.	Mock/A Check
Physics Variables Initialized	`Llama()` called	`self.velocity_y` is initialized to 0. `self.is_jumping` is initialized to False.	Attribut Check
Initial Position Calculation & Assignment	`Llama()` called	`self.initial_pos` tuple equals `(constants.PLAYER_HORIZONTAL_POSITION, constants.GROUND_Y - self.rect.height)`. `self.rect.bottomleft` is set to the value of `self.initial_pos` after calculation.	Attribut Check

Test Results

Test 01

Test Results - llama_init - test_01.html

Passed 8/8 tests

Update Player State (`update`)

Component Planning

Update Player State (`update`)

- [ ] Apply the effect of gravity to the player's vertical speed.
- [ ] Change the player's vertical position based on its current speed.
- [ ] Check if the player has landed on or fallen below the ground.
- [ ] If on the ground, stop downward movement and reset vertical speed.
- [ ] (Optional) Change the player's visual appearance based on state (jumping/running).

- [ ] (Optional) Update the collision mask if the visual appearance changed.

Test Plan

Test Case	Input / Conditions	Expected Output	Test Type
Apply Gravity	Called while `velocity_y` is 0 or positive	`self.velocity_y` increases by `GRAVITY`. `self.rect.y` increases by `int(self.velocity_y)`.	Expected
Apply Gravity (Moving Up)	Called while `velocity_y` is negative	`self.velocity_y` increases (becomes less negative). `self.rect.y` changes according to `int(self.velocity_y)`.	Expected
Ground Collision Check (On Ground)	`self.rect.bottom` is exactly `constants.GROUND_Y`	`self.rect.bottom` remains `constants.GROUND_Y`, `self.velocity_y` becomes 0, `self.is_jumping` becomes False.	Boundary
Ground Collision Check (Below)	`self.rect.bottom` > `constants.GROUND_Y` (e.g., +1px)	`self.rect.bottom` corrected to `constants.GROUND_Y`, `self.velocity_y` becomes 0, `self.is_jumping` becomes False.	Boundary
No Ground Collision (Mid-Air)	`self.rect.bottom` < `constants.GROUND_Y`	`self.rect.bottom` changes based on `int(velocity_y)`. `self.velocity_y` changes due to gravity. `is_jumping` state remains unchanged.	Expected

Test Results

Test 01

Test Results - llama_update - test_01.html

Passed 6/6 tests

Perform Jump (`jump`)

Component Planning

Perform Jump (`jump`)

- [ ] Check if the player is currently on the ground.
- [ ] If on the ground, give the player an upward vertical speed boost.
- [ ] (Optional) Change the player's visual appearance to jumping state.

Trialling

I trialled two different methods of performing a jump.

Method 1:

This method involved moving the llama up in certain increments, and then back down in certain increments. This method resulted in the jump being performed, but it looked jerky and unrealistic.

Method 2:

This method involved using basic gravity physics to model the jump. When the jump key is pressed, the llama's velocity is set to a value (taken from `constants.py`), and every frame the velocity is changed by the gravity value defined in `constants.py`. Once the llama's vertical position is at the ground level, the velocity is set to 0. This results in a natural looking jump, which can easily be adjusted by changing the respective values.

Ultimately I chose method two, as it looks better and is more adjustable than method one.

Test Plan

Test Case	Input / Conditions	Expected Output	Test Type
Jump When Not Jumping	`self.is_jumping` is False	`self.velocity_y` becomes `constants.JUMP_SPEED`, `self.is_jumping` becomes True.	Expected
Attempt Jump When Already Jumping	`self.is_jumping` is True	No change to `self.velocity_y` or `self.is_jumping`.	Expected

Test Results

Test 01

Test Results - llama_jump - test_01.html

Passed 3/3 tests

Reset Player (`reset`)

Component Planning

Reset Player (`reset`)

- [ ] Move the player back to its initial starting position.
- [ ] Reset the player's vertical speed to zero.
- [ ] (Optional) Reset the player's visual appearance to the default (running) state.

Test Plan

Test Case	Input / Conditions	Expected Output	Test Type
Reset After Jump	Called after Llama has jumped	`rect.bottomleft` returns to `initial_pos`. `velocity_y` becomes 0. `is_jumping` becomes False.	Expected
Reset While Moving	Called while Llama is mid-air	`rect.bottomleft` returns to `initial_pos`. `velocity_y` becomes 0. `is_jumping` becomes False.	Expected
Reset From Ground	Called while Llama is on ground	`rect.bottomleft` remains at `initial_pos`. `velocity_y` remains 0. `is_jumping` remains False.	Expected

Test Results

Test 01

Test Results - llama_reset - test_01.html

Failed 3/3 tests.
This is because `Llama.reset` sets `self.initial_pos` to the correct coordinates for the top left of the sprite, then sets `self.rect.bottomleft` to `self.initial_pos`

```
self.initial_pos = (
    constants.PLAYER_HORIZONTAL_POSITION,
    constants.GROUND_Y - self.rect.height,
)
self.rect.bottomleft = self.initial_pos
```

To fix this, I changed the code to set the *top* left corner to `self.inital_pos`.

```
self.initial_pos = (
    constants.PLAYER_HORIZONTAL_POSITION,
    constants.GROUND_Y - self.rect.height,
)
self.rect.topleft = self.initial_pos
```

Test 02

Test Results - llama_reset - test_02.html

Passed 3/3 tests

Obstacle Class - Llama Game Decomposition

Setup Obstacle (`__init__`)

Component Planning

Setup Obstacle (`__init__`)

- [ ] Initialize the base Sprite features.
- [ ] Load the obstacle's visual appearance (image or shape).
- [ ] (Optional) Randomly choose which type of obstacle appearance to use.
- [ ] Set the obstacle's visual appearance.
- [ ] Get the rectangle representing the obstacle's position and size.
- [ ] Set the obstacle's starting position (off-screen to the right).
- [ ] Store the speed at which the obstacle should move.
- [ ] Create a precise outline (mask) for collision detection.

Test Plan

Test Case	Input / Conditions	Expected Output / Checks	Test Type
Call Superclass Init	`Obstacle(speed)` called	`pygame.sprite.Sprite.__init__(self)` is called exactly once.	Mock Chec
Image Load Success Path	`pygame.image.load` succeeds	`pygame.image.load` called with `constants.OBSTACLE_IMAGE`. `convert_alpha()` called on result. `self.image` assigned the converted surface.	Mock/Attr Check
Image Load Failure Path (Fallback)	`pygame.image.load` raises `Exception`	Exception caught. `pygame.Surface` created with `[25, 50]`. `fill()` called with `constants.GREEN`. `self.image` assigned the fallback surface.	Error Handl
Rect and Mask Creation (Success/Failure)	`Obstacle()` called (image loaded OR fallback used)	`self.image.get_rect()` called. `self.rect` assigned the result. `pygame.mask.from_surface` called with `self.image`. `self.mask` assigned the result.	Mock/Attr Check
Initial Position (Off-Screen X, Ground Y)	`Obstacle(speed)` called	`random.randint(50, 200)` is called. `self.rect.left` is $\geq$ `constants.WINDOW_WIDTH + 50`. `self.rect.bottom` is `constants.GROUND_Y`.	Expected/R
Speed Assignment	`Obstacle(10)` called	`self.speed` attribute is set to 10.	Attribute Ch

Test Results

Test 01

Test Results - obstacle_init - test_01.html

Passed 6/6 tests

Update Obstacle State (`update`)

Component Planning

Update Obstacle State (`update`)

- [ ] Move the obstacle horizontally to the left based on its speed.
- [ ] Check if the obstacle has moved completely off the left side of the screen.
- [ ] If off-screen, remove the obstacle from the game.

Test Plan

Test Case	Input / Conditions	Expected Output	Test Type
Move Left	Called once	`self.rect.x` decreases by `self.speed`.	Expected
Move Left Repeatedly	Called multiple times	Obstacle moves steadily left across the screen.	Expected
Off-Screen Check (On Screen)	`self.rect.right` >= 0	Obstacle remains in its sprite groups. `kill()` is not called.	Expected
Off-Screen Check (Boundary)	`self.rect.right` becomes < 0	`self.kill()` is called (verified by checking sprite group membership afterwards).	Boundary
Zero Speed Obstacle	`self.speed` is 0	`self.rect.x` does not change. Obstacle never goes off-screen left via movement.	Edge Case

Test Results

Test 01

Test Results - obstacle_update - test_01.html

Passed 5/5 tests

Scoreboard Class - Llama Game Decomposition

Setup Scoreboard (`__init__`)

Component Planning

Setup Scoreboard (`__init__`)

- [ ] Store the desired position and color for the score display.
- [ ] Load or prepare the font for rendering text.
- [ ] Set the initial score value to zero.
- [ ] Prepare variables to hold the rendered score text image and its position.
- [ ] Create the initial score text image (e.g., "Score: 0").

Test Plan

Test Case	Input / Conditions	Expected Output / Checks	Test Type
Initialization (Defaults)	`Scoreboard()` called	`self.x` =10, `self.y` =10, `self.color` =constants.BLACK, `self.score` =0. `pygame.font.SysFont` called with `(None, 36)`. `self.font` is a Font object. `_render_text` called once.	Attribute/Mock Check
Initialization (Custom Args)	`Scoreboard(x=50, y=60, font_size=48, color=RED)`	`self.x` =50, `self.y` =60, `self.color` =constants.RED, `self.score` =0. `pygame.font.SysFont` called with `(None, 48)`. `self.font` is a Font object. `_render_text` called once.	Attribute/Mock Check
Font Object Creation	`Scoreboard()` called	`pygame.font.SysFont` called once with `(None, 36)`. `self.font` holds the returned Font object.	Mock Check
`_render_text` Call Verification	`Scoreboard()` called	`self._render_text` method is called exactly once during initialization.	Mock Check

Test Results

Test 01

Test Results - scoreboard_init - test_01.html

Passed 4/4 tests

Update Score (`update`)

Component Planning

Update Score (`update`)

- [ ] Calculate the current score based on how long the game has been running.
- [ ] Check if the calculated score is different from the currently displayed score.
- [ ] If the score has changed, store the new score value.
- [ ] If the score has changed, create a new text image for the updated score.
- [ ] If the score has changed, update the position rectangle for the new text image.

Test Plan

Test Plan: Scoreboard.update

Test Case	Input / Conditions	Expected Output / Checks	Test Type
No Score Change (< 10ms)	`current_time` - `game_start_time` < 10ms (e.g., 9ms)	`self.score` remains unchanged (e.g., 0). `font.render` is *not* called again. `self.image` remains unchanged.	Expected/Mock
Score Increase (>= 10ms)	`current_time` - `game_start_time` >= 10ms (e.g., 10ms)	`self.score` increases by 1. `font.render` is called once, `self.image` updates with the new score text.	Expected/Mock
Boundary Check (Exactly 10ms)	`current_time` - `game_start_time` == 10ms	`self.score` increases to 1 (from 0). `font.render` is called once.	Boundary/Mock
Multiple Interval Update	`current_time` - `game_start_time` = 55ms	`self.score` becomes 5 (55 // 10). `font.render` is called once.	Expected/Mock
No Score Change (Score Stays)	Call update twice with same `new_score` (e.g., 55ms then 59ms)	First call: score becomes 5, `font.render` called. Second call: score remains 5, `font.render` *not* called again.	State/Mock Check
Large Time Values	`current_time` - `game_start_time` = 1,234,567ms	`self.score` becomes 123456 (1234567 // 10). `font.render` is called.	Boundary/Mock
Time Reset (Conceptual)	`game_start_time` changes between calls	Score calculation correctly uses the *new* `game_start_time` passed in the arguments for subsequent calls.	Dependency Check
Negative Time Difference	`current_time` < `game_start_time` (e.g., `current` =0, `start` =10)	`new_score` calculates to -1. `self.score` becomes -1. `font.render` is called (as score changed from 0).	Robustness/Edge

Test Case	Input / Conditions	Expected Output / Checks	Test Type
Verify Render Arguments	Score changes (e.g., 0 → 1)	`font.render` called with `f"Score: {new_score}"`, `True`, `self.color`. `get_rect` called on the result.	Mock Check

Test Results

Test 01

Test Results - scoreboard_update - test_01.html

Passed 9/9 tests

Draw Score (`draw`)

Component Planning

Draw Score (`draw`)

- [ ] Draw the current score text image onto the main game screen.

Test Plan

Test Case	Input / Conditions	Expected Output / Checks	Test Type
Draws Current Score Image	Called with valid `screen` surface	`screen.blit` is called once with the current `self.image` and `self.rect` attributes.	State/Mock Check
Draws Updated Score Image	Called after `update` has changed `self.image` / `self.rect`	`screen.blit` is called once with the *updated* `self.image` and `self.rect` attributes.	State/Mock Check

Test Results

Test 01

Test Results - scoreboard_draw - test_01.html

Passed 2/2 tests

Reset Score (`reset`)

Component Planning

Reset Score (`reset`)

- [ ] Set the score value back to zero.
- [ ] Create the text image for the zero score.
- [ ] Update the position rectangle for the zero score text.

Test Plan

Test Case	Input / Conditions	Expected Output / Checks	Test Type
Reset Score to Zero	Called after score increased	`self.score` becomes 0. `self.image` surface is updated to show "Score: 0". `self.rect` position is updated correctly.	State Check
Reset When Already Zero	Called when `self.score` is 0	`self.score` remains 0. `self.image` surface is updated to show "Score: 0". `self.rect` position is updated correctly.	State Check
Verify Font Render Call Args	Called (either case above)	`self.font.render` is called with `f"Score: 0"` , `True` , and `self.color` . `self.image.get_rect` called with `topleft` .	Mock Check

Test Results

Test 01

Test Results - scoreboard_reset - test_01.html

Passed 3/3 tests

Assembled outcome - Testing

Test 01

When run the game returns an error:

```
    self.clock.tick_busy_loop(constants.FPS)
    ~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^
TypeError: descriptor 'tick_busy_loop' for 'pygame.time.Clock' objects doesn't apply to a 'int' object
```

The issue was when setting up the game clock, I used `self.clock = pygame.time.Clock` . This assigns the class `pygame.time.Clock` to `self.clock` instead of assigning an instance of `pygame.time.Clock` . To fix this I added parentheses to the end of the line:

```
self.clock = pygame.time.Clock()
```

This fixed the error

Test 02

When running the game, this screen is shown:

Score increases but nothing else on the display changes. The issue was in `constants.py`, as I had left some of the values like `JUMP_HEIGHT` and `GRAVITY` at `0`. When setting these to realistic values the game actually ran properly.

Test 03/Trial

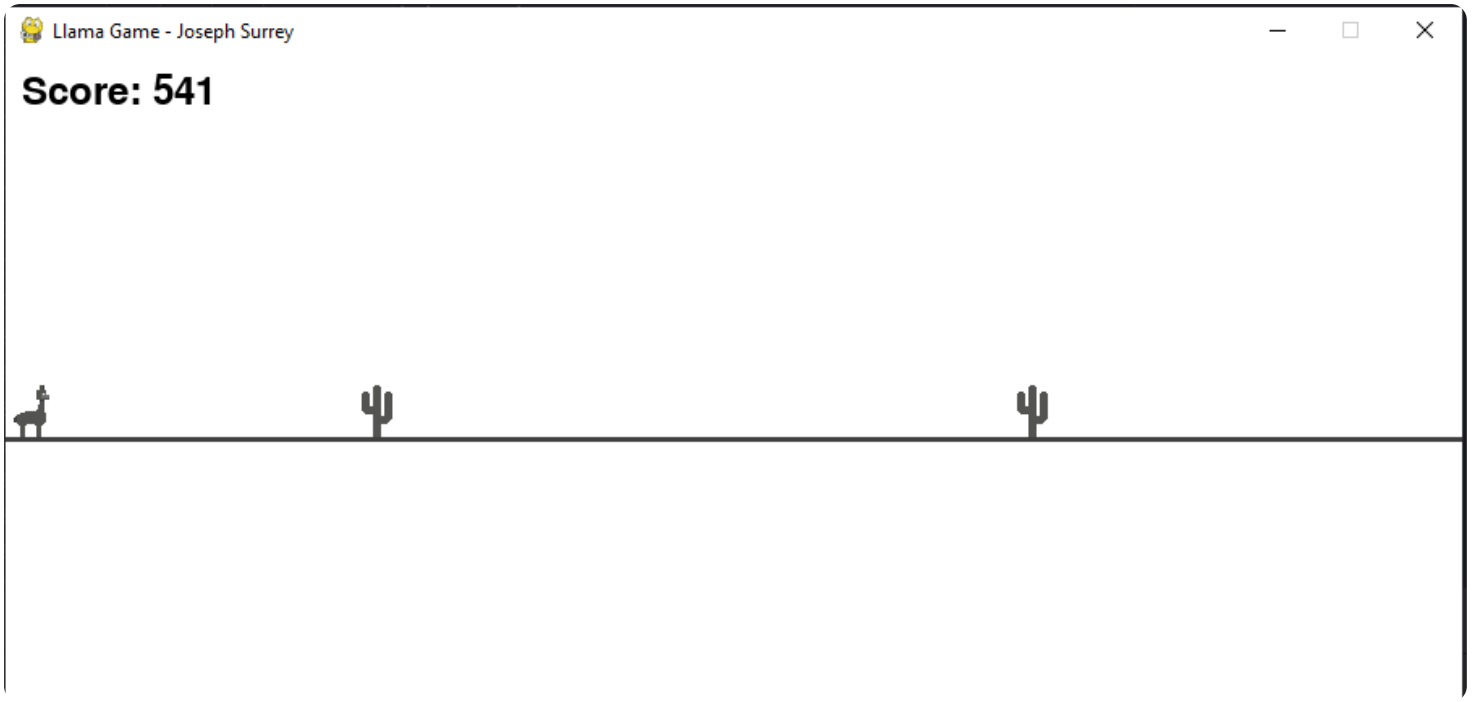
The next thing to fix was the background image and the spawn height of the llama. The background image is not scaled to the size of the window, and the llama is spawning at the incorrect height. After making some changes to the way the image is loaded, it now correctly fits the screen. I trialled two different ways of doing this.

Method 1:

I set the window size to the exact dimensions of the image. This resulted in the image matching perfectly with the window.

Method 2:

I calculated the aspect ratio of the image, and then scaled it up so that the aspect ration remained the same but the height matched the window height (defined in constants). This means that more complicated backgrounds and different sized windows can be used while keeping the size and scale accurate and aesthetic. I chose to use this option in my outcome.



The image now scales correctly and the llama stands on the ground.

Test 04

The next issue is with restarting the game. After a restart, the score doesn't reset and the llama changes position.

Note

Image taken directly after restart (no obstacles have spawned yet)

This is due to using the wrong variable name when resetting the score. I used:

```
# Reset the start time for the new game
self.start_time_ticks = pygame.time.get_ticks()
```

When I should have been using:

```
# Reset the start time for the new game
self.start_time = pygame.time.get_ticks()
```

This fixed the issue of the incorrect score.

The issue with the llama changing position wasn't actually because after resetting it was in the wrong place, it was because the llama was spawning in the wrong place when the program is run. To fix this, I run `llama.reset()` after creating the llama, which puts it into the right position.

The game now runs as expected.

Assembled Outcome Integration Tests (pytest)

Game Integration Tests

Test Case	Scenario	Key Interactions Tested	Expected Outcome/State Checks
Game Initialization	`Game()` is instantiated.	`__init__` execution, Pygame module init, screen setup, clock creation, sprite group setup, Llama/Scoreboard creation.	Game object created, `running` =True, `game_over` =False, `llama` instance present, `scoreboard` instance present, `all_sprites` contains llama, `obstacles` empty, `start_time` set, obstacle timer scheduled.
Basic Gameplay Loop (Tick)	Simulate one frame update during gameplay.	`_update` → `all_sprites.update`, `scoreboard.update`, `_check_collisions`.	`llama.update` called, `scoreboard.update` called (score might increase), `_check_collisions` called, `game_over` remains False.
Player Jump Event	Simulate SPACE key press during gameplay.	`_handle_events` → `llama.jump`.	`llama.jump` method is invoked.

Test Case	Scenario	Key Interactions Tested	Expected Outcome/State Checks
Obstacle Spawn Event	Simulate `OBSTACLE_SPAWN_EVENT` during gameplay.	`_handle_events` → `_spawn_obstacle` → `Obstacle` creation, adding to groups.	New `Obstacle` added to `obstacles` and `all_sprites` groups.
Collision Detection	Simulate llama colliding with an obstacle during gameplay.	`_update` → `_check_collisions` → `pygame.sprite.spritecollide` returns collision.	`game_over` becomes True, obstacle spawn timer is cancelled (`set_timer` with 0).
Game Over State (No Input)	Game is in the `game_over` state. Simulate one frame update.	`_update` logic bypass, `_draw` logic for game over screen.	`all_sprites.update`, `scoreboard.update`, `_check_collisions` are *not* called. Drawing methods for "Game Over" text, score, and instructions are called.
Restart Event	Simulate 'R' key press during game over.	`_handle_events` → `_reset_game`.	`game_over` becomes False, `obstacles` cleared, `all_sprites` reset (only llama), `llama.reset` called, `scoreboard.reset` called, `start_time` updated, obstacle timer restarted.
Quit Event (Game Over)	Simulate 'Q' key press during game over.	`_handle_events` → sets `self.running` to False.	`game.running` becomes False.
Quit Event (Window Close)	Simulate `pygame.QUIT` event (at any time).	`_handle_events` → sets `self.running` to False.	`game.running` becomes False.
Drawing Cycle (Playing)	Simulate drawing during gameplay.	`_draw` → `screen.fill`, `screen.blit` (ground), `all_sprites.draw`, `scoreboard.draw`, `pygame.display.flip`.	Corresponding mock methods (`fill`, `blit`, `draw`, `flip`) are called with expected arguments.
Drawing Cycle (Game Over)	Simulate drawing during game over.	`_draw` → `screen.fill`, `screen.blit` (ground), `all_sprites.draw`, `scoreboard.draw`, Game Over text rendering/blit.	Corresponding mock methods are called, including rendering/blitting of "Game Over", final score, and instructions.

Test Results

Test 01

Test Results - game_intergration - test_01.html

Passed 11/11 integration tests

Final Discussion

Addressing relevant implications

- **Functionality** By finding and fixing bugs in individual parts and how they worked together, I made sure the game runs, the player can jump, obstacles appear, collisions end the game, and restarting works as intended.
- **Usability** I planned simple controls (Space/Up to jump, R to restart, Q to quit). Testing confirmed these controls worked only when they were supposed to. Clear on-screen instructions tell the player what to do. The smoother jump (from trialling) also makes it easier to control the llama.
- **Aesthetics** I planned to use images instead of plain shapes. Testing made sure the images loaded. Trialling helped me choose the best way to scale the background and make the jump look natural, improving the game's visual appeal.
- **Social** I thought about whether a simple game like this could be harmful (like being too addictive). I decided the risk was very low for this basic game, so no special features were needed to prevent problems.

Complex processes

Planning the Game:
First, I planned how the game would work. I decided on the different parts (like the `Game`, `Llama`, `Obstacle` classes) and what each part should do. I also made a `constants.py` file to keep important settings (like window size, speed, image names) in one place. This plan gave me a clear guide to follow.

Testing the Parts:
As I built each part, I tested it carefully. These tests helped me find problems early:

- The tests for `constants.py` showed I had written the image file locations incorrectly, which stopped the images from loading.
- A test for setting up the `Game` found a mistake in how I started the game clock, which would have crashed the game.
- Testing the `Llama`'s reset function showed the llama wasn't starting in the right spot

Fixing these bugs found during testing made the game work correctly and stopped it from crashing.

Trialling

- **Jumping:** I tried two ways for the llama to jump. The first way looked jerky. The second way used simple physics (gravity and speed) and looked much smoother and more natural, so I chose that one.
- **Background:** I decided to scale the background to fit the screen, rather than scale the screen to fit the background to make it easy to use other backgrounds in the future.

The combination of these three processes helped me to develop an outcome that was of a very high quality.