HSC Software Engineering

2025

Personal Project - ‘Stellar Odyssey’

Vincent Hudaja

1.0 Identifying & Defining

→ 1.1 Problem Statement

Young learners often struggle to engage in educational content through traditional learning methods. This project addresses the need for a more interactive and enjoyable way to explore the Solar System through an educational game that encourages curiosity and active learning for users of all ages, particularly children.

→ 1.2 Project Purpose & Boundaries

The goal is to create an interactive 2D game that educates users on the Solar System in an engaging way. The game will allow users to explore celestial bodies, learn facts and collect items through gameplay. Within the scope, a 2D Solar System simulation with interactive features and reliable user data storage is achievable. However realistic 3D graphics, multiplayer functionality and complex physics simulations are restricted due to time and technical limitations.

→ 1.3 Stakeholder Requirements

For educational users, the game must be fun and an intuitive learning experience. For educators and other academically based organisations, software that improves engagement and understanding is required. Overall accessibility, informativity and engagement is key across all stakeholders.

→ 1.4 Functional Requirements

- Player movement using keyboard input
- Collision detection with celestial bodies and interactive objects
- Switching between views (menu, Solar System, surfaces etc.)
- Displaying fact files and inventory
- Saving user data locally (collected items, credentials)

→ 1.5 Non-Functional Requirements

- Consistent 60 FPS during runtime
- Simple, age-friendly interface and controls
- Clean visuals and consistent design across UI and sprites
- Basic sound for immersions
- Portability

→ 1.6 Constraints

This project is limited to a 10 week development period. Additionally, only free resources and tools are utilised to develop my game. My game will be technically limited by 2D graphics, no built-in UI framework and data persistence through JSON without the use of external databases.

2.0 Research & Planning

→ 2.1 Development Methodology

I used an Agile approach, breaking development into sprints. This allowed me to iteratively plan, develop and test features such as player movement, interactions, surfaces and inventory. Agile suited my project well due to the need for continuous improvements based on tests and feedback on several key components of my code from interactions to data handling.

→ 2.2 Tools & Technology

- **Language:** Python 3
- **IDE:** Visual Studio Code
- **Libraries:**
    - **Standard:** os, sys, json, re, math, random
    - **Third Party:** pygame (game window, graphics, user input, UI, etc.), bcrypt (secure data handling)
- **Assets:** Custom sprites, sounds and surfaces for celestial bodies created or sources from free online resources (google, piskel)

→ 2.3 Gantt Chart/Timeline

Stage	Start Week	End Week	Estimated Hours	Milestone
Planning Game	1	2	7	Project plan completed with thorough idea on gameplay and overall goal
Create Main Game Graphics	2	3	3	Main game sprites, buttons and other visual features created
Code Game Menu & Basic Gameplay	2	3	5	Home menu screen and basic gameplay in-game completed
Test + Refine	3	4	2	Tests for display functionality and

Display & GUI Code				GUI completed
Code Interactions for Player Sprite & Other In Game Objects	3	4	3	Interactions between player and other sprites in game working
Test + Refine Interactions & Collisions Code for Sprites	4	5	2	Tests for interaction and collisions for players and in-game sprites completed
Code 'Fact Files' For Each Celestial Body	4	5	2	Individual/unique fact files for each celestial body completed
Create Graphics for Celestial Surfaces	5	6	2	Unique graphics created for different celestial bodies
Code Separate Surfaces for Each Celestial Body	5	7	7	Code and functionality for surface interactivity and features completed including collectable items
Test + Refine Surface Code for Celestial Bodies	7	8	3	Tests for Fact File and Surface functionality completed
Code Player Inventory Functionality	7	8	3	Player inventory completed with item collection and storage functionality
Code Data Saving & Security	8	9	2	Working data storage through login system, functionality of saving user progress
Test + Refine Code Data Saving & Security	8	9	2	Tests for data saving and security completed
Clean Up Code + Small Additions	9	10	2	Cleaned up code for readability and maintainability, smaller additions for convenience

Evaluate Code	9	10	2	Evaluate the final completed game code addressing goals, functionality and convenience

Week →	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Planning Game										
Create Main Game Graphics										
Code Game Menu & Basic Gameplay										
Test + Refine Display & GUI Code										
Code Interactions for Player Sprite & Other In Game Objects										
Test + Refine Interactions & Collisions Code for Sprites										
Code 'Fact Files' For Each Celestial Body										
Create Graphics for Celestial Surfaces										
Code Separate Surfaces for Each Celestial Body										
Test + Refine Surface Code for Celestial Bodies										
Code Player Inventory Functionality										
Code Data										

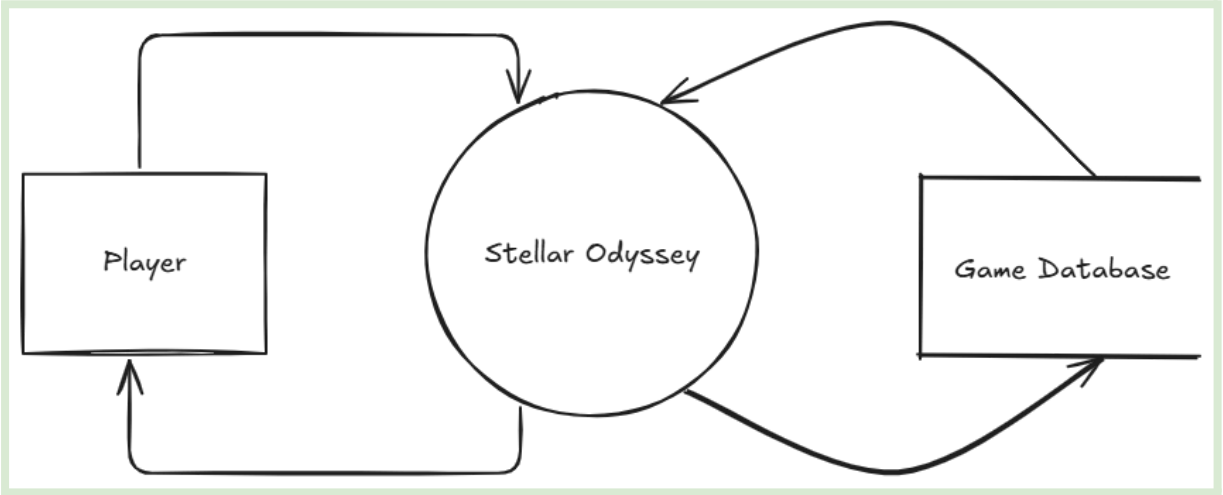
Saving & Security										
Test + Refine Code Data Saving & Security										
Clean Up Code + Small Additions										
Evaluate Code										

→ 2.4 Communication Plan

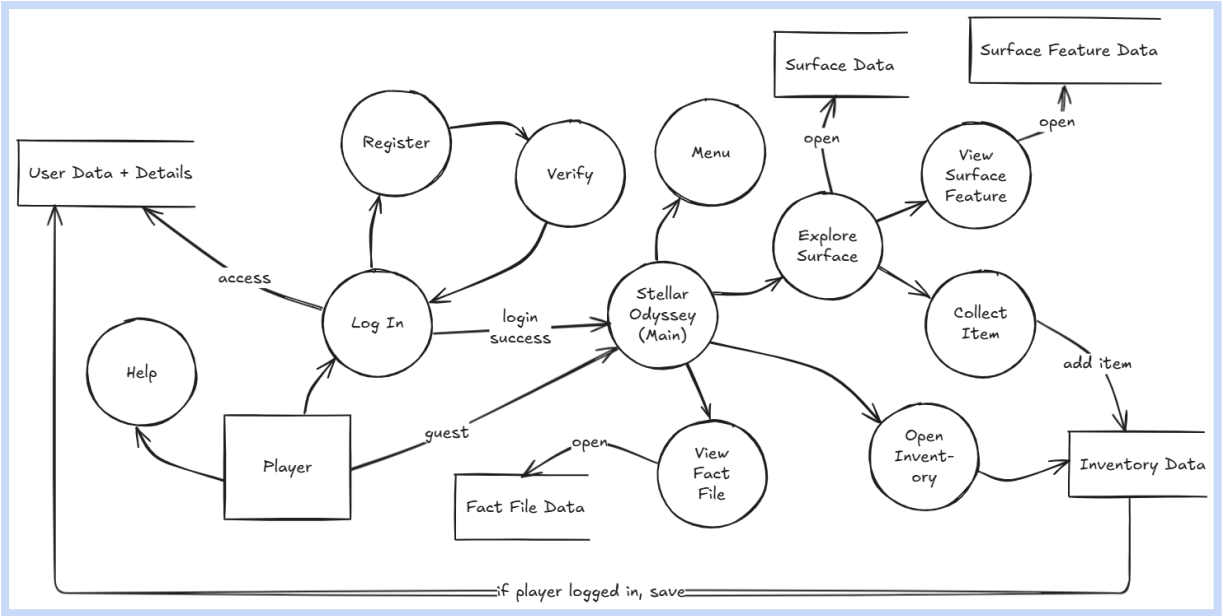
Early development proposals and checkpoints will be submitted for review and guidance. Additionally, informal testing and feedback sessions through my peers will assist in refining gameplay, usability and educational clarity.

3.0 System Design

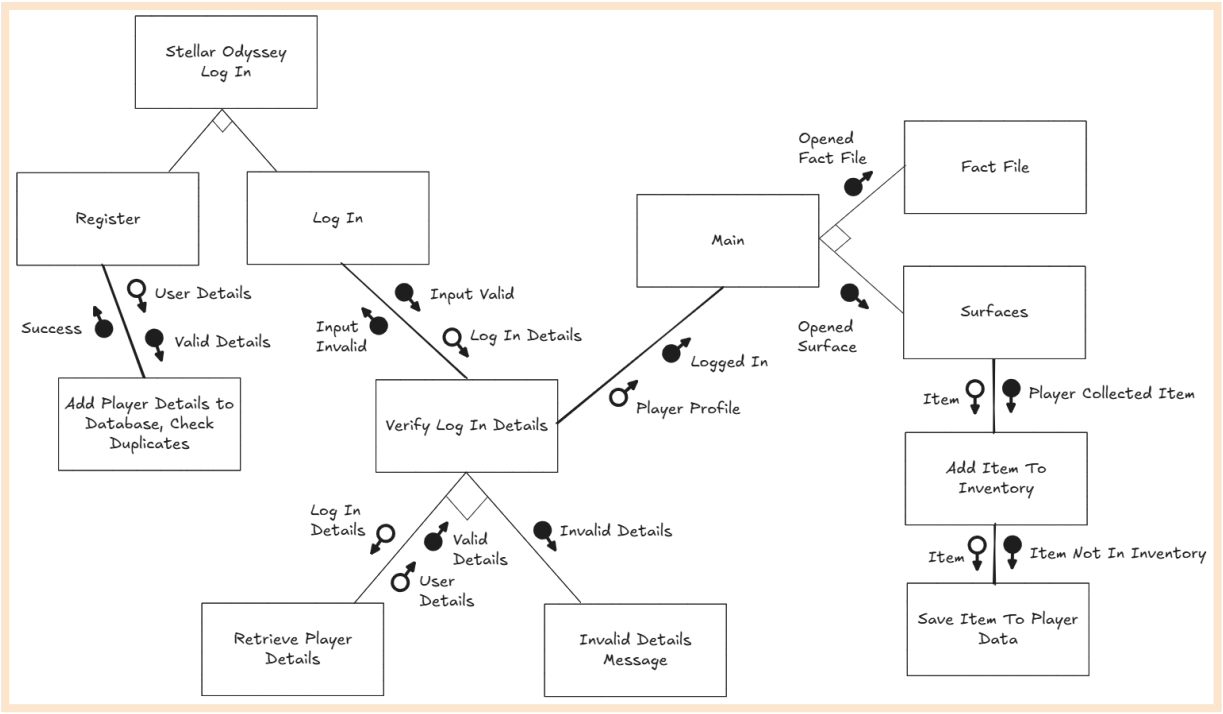
→ 3.1 Context Diagram (Level 0 Data Flow)



→ 3.2 Level 1 Data Flow Diagram



→ 3.3 Structure Chart



→ 3.4 IPO Chart

Input	Process	Output
User Actions (Keyboard/Mouse), System Events	Initialises game, runs main loop, establishes modules and interactions	Game window updates, state transitions
Username, Password	Validates input, checks	Login result,

	credentials, registers/logs in player	error/success message, loaded user data
File Paths (Images, Fonts, Sounds)	Loads and scales game assets	Asset variables
Pygame Events, Player State, Keypresses	Processes inputs, updates state (movement, collisions), handles transitions	Updated game logic, changes to states
Mouse Position	Renders menu with scrolling background effect (parallax), hover effect for buttons	Menu screen rendered on display
Camera Position, Celestial Body States, Player Position	Render background, sun, planets, collectibles and player	Updated game view
Key Presses (WASD), Delta Time	Applies movement friction (drift), updates player velocity and position	New player position and velocity
Active Celestial Body, Mouse Position, Scroll Events	Displays celestial body data, handles scrolling	Rendered fact file UI with text and visuals
Celestial Body Name, Player Position, Mouse Position	Loads surface graphics, checks feature hover/collision, draws labels	Surface rendered, features labelled, fact boxes opened/closed
Player Position, Opened Facts, Current Inventory	Checks if collectible is obtained or not, handles collection	Item added to inventory, sound played
Items List, Draw Request	Loads images of items, arranges grid, draws icons	Inventory grid shown with item images
Keystrokes, Mouse Events	Renders typed characters, reveals/hides password, scrolls for longer text	Username/password input display
Username, Inventory Items	Writes/reads inventory to/from file	File storage of player progress
Orbital Parameters, Delta Time	Updates position in orbit, checks collision with player	Celestial body position updated, hover state set

→ 3.5 Data Dictionary

Name	Data Type	Size/Format	Description	Example Value	Constraints/ Validation Rules
current_user	str / None	≤ 20 chars	Logged-in user's username	"vin" / None	Alphanumeric only, required to access save/load
player_pos	list[int]	[x: 0–7000, y: 0–7000]	Player's world coordinates	[3500, 3600]	Must remain within WORLD_WIDTH × WORLD_HEIGHT
player_vel	list[float]	[vx, vy] floats	Player's current velocity vector	[2.5, -1.0]	Clamped by max_speed = 10
zoom	float	1 decimal place	Current zoom level	1.0	Must be between zoom_min = 0.5 and zoom_max = 2.0
planets	list[Planet]	9 elements	All orbiting planets	["Mercury", "Venus", "Earth", ...]	Each must be unique by name; initialised from planet_data
FactFile	class	Object with 10+ attributes	Celestial body fact data display system	FactFile("Mars", ...)	All fields required; composition dict optional
SurfaceFeatureSet	class	List of features	Feature UI manager for surfaces	SurfaceFeatureSet([{"name": "crater", ...}])	Each feature must have unique name per celestial body
Collectible	class	8 instances total	Represents surface collectible	Collectible("Leaf", "Earth", ...)	Each must be linked to valid celestial body

			items		and feature unlock states
player_inventory	Inventory class	Max 10 items	Stores collected item names	["leaf", "plasma", "basalt"]	No duplicates; only allows items from pre-defined asset folder
factfile_data	dict	10 keys (celestial body names)	Factual data about celestial bodies	{"Mars": {"mass": "...", ...}}	Must match celestial body names exactly
surface_feature_data	dict	10 keys (celestial body names)	Defines surface features and positions	{"Venus": [{"name": "tessera", ...}]}	Each pos must be in (x,y) within screen dimensions
feature_facts_data	dict	20+ keys (feature names)	Fact text for each surface feature	{"lava_plains": "These are flat areas..."}	Must be string; wrapped via wrap_text() for display
input_username	InputBox class	300px wide text box	Username entry field	"cosmic_player"	≤ 20 chars, must be alphanumeric only
input_password	InputBox class	300px wide password box	Password entry field	"supernova123"	≤ 20 chars, must be alphanumeric; shows * unless revealed
login_mode	str	"sign_in" or "sign_up"	Current auth screen mode	"sign_up"	Must match enum values only
movement_tip_alpha	int	0–255	Transparency of movement instruction	127	Computed dynamically, clamped
surface_images	dict[str, Surface]	Up to 10 entries	Preloaded Pygame surfaces for celestial	{"Mars": <Surface object>}	Image file must exist for each key

			body		
collectibles	list[Collectible]	8 total objects	Surface item definitions	[Collectible("Plasma", ...), ...]	Unique per celestial body; one collectible per surface
zoom_min, zoom_max	float	Single precision	Zoom boundaries	0.5, 2.0	zoom_min < zoom_max, must be > 0
in_game, in_menu, (etc.)	bool	1 bit	State flags	True / False	Only one main screen state should be true at a time
fact_files	dict[str, FactFile]	9–10 entries	Mapping of celestial body names to factfile instances	{"Venus": FactFile(...), ...}	Keys must match entries in factfile_data
active_planet	Planet / None	Object or null	Planet currently focused on	Planet("Earth", ...)	Must exist in planets; optional type with .name attribute
sun_center	tuple[int, int]	(3500, 3500) fixed	World coordinate center of the Sun	(3500, 3500)	Immutable after init
bg_tile, bg_image	Surface	7000x7000 composed surface	Background rendered behind the world	<Surface>	Requires loaded image to scale to tiling dimensions
surface_player_pos	list[int]	Within screen (0–1280, 0–720)	Position on celestial body surface screen	[450, 360]	Clamped to screen bounds
surface_player_vel	list[float]	Velocity vector	Motion on surface	[1.2, -0.8]	Controlled via W/A/S/D keys, limited by surface max speed

show_arrows, show_tip, (etc.)	bool	True / False	UI-related indicators	True	Used for guidance; automatically hidden by timers
*_fact_opened	dict[str, bool]	2–4 per celestial body	Flags for visited features	{"caloris_basin": True}	Must be updated on fact view; False by default
*_collected	bool	True / False	Indicates if item is already collected	False/True	Defaults to False; set True upon collection and saved if logged in

→ 3.6 UML Class Diagram

4.0 Producing & Implementing

→ 4.1 Development Process

I used an iterative Agile approach, constructing my game feature by feature. I started with basic mechanics, player movement and rendering celestial bodies, then gradually added more complex features including surface exploration, fact files, collectibles, inventory and saveable user data. Each feature was developed in a sprint, tested and refined before proceeding to the next. This ensured consistent progress and enabled flexible adjustments based on testing outcomes or feedback.

→ 4.2 Key Features Developed

Surface Exploration
Players can explore unique surfaces for each celestial body. This promotes interactive learning by allowing players to engage directly with unique landscapes.

Fact Files/Boxes
Each surface feature contains educational fact boxes with scientific information. These support the educational objective of the game by delivering content in an interactive, engaging format.

Collectable Items
Once players view fact boxes in certain surfaces, they unlock a unique collectible. This mechanic rewards curiosity and encourages thorough exploration.

Inventory System
Collected items are stored in an in-game inventory. This provides players with a sense of achievement and reinforces the game's educational progression.

User Login/Sign Up System
My game includes secure login functionality with encrypted passwords and user-specific progress storage through JSON. This ensures users can resume their sessions and retain collected items.

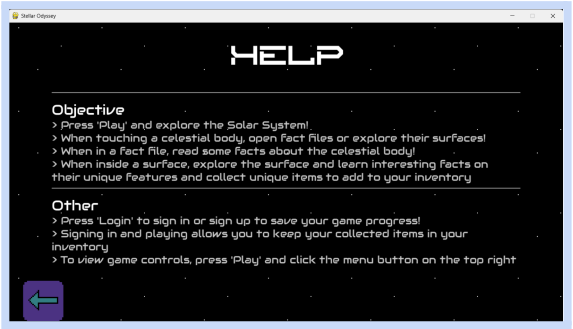
User Interface
A clean, intuitive interface with tips and consistent visuals helps guide players through the game. It enhances usability, especially for younger players.

→ 4.3 Screenshots of Interface

Home Menu: Allows access to play, login and help

Login Page: Allows player log in

Sign Up Page: Allows registration of new players

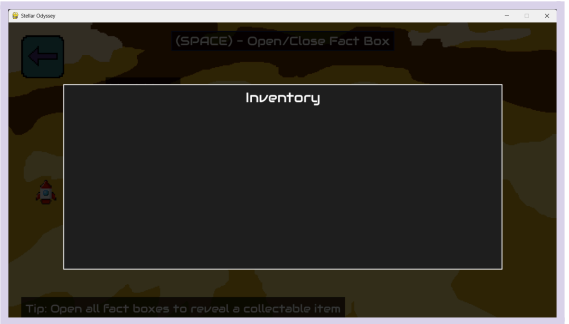


Help Page: Provides information on gameplay/objectives

Main Game: Main game interface for players to explore

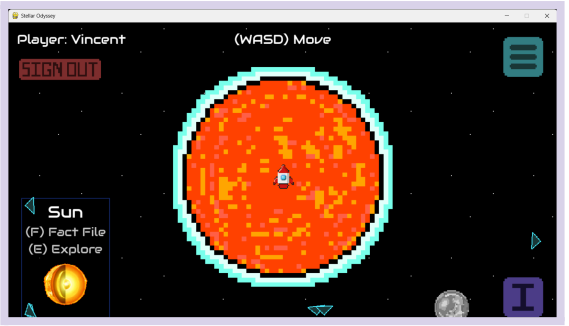
Fact File: Page of factual information

Surface: Explorable surface of features

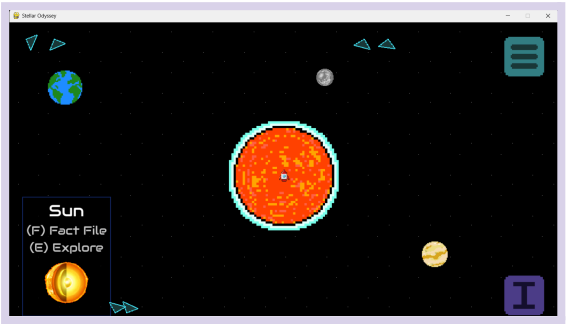


Surface Inventory: Inventory UI on surface, allows player to view collected items

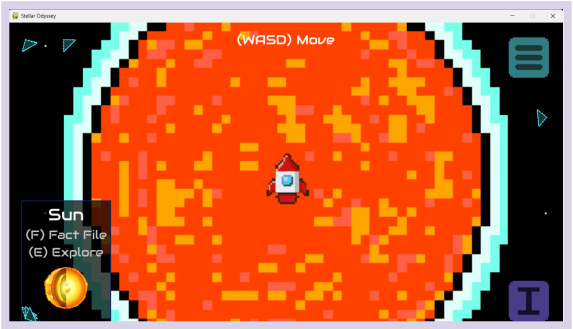
Main Inventory: Main inventory UI, allows player to view collected items



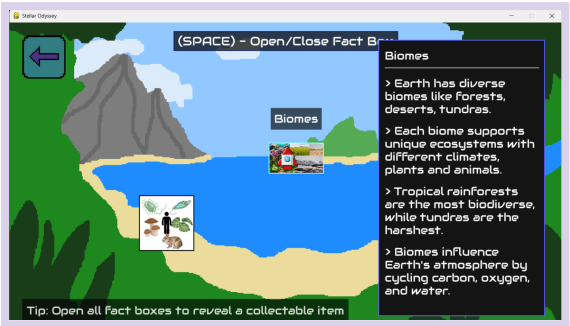
Logged-In Main Game: Main game interface with player profile



Zoom Out: Zoomed out UI

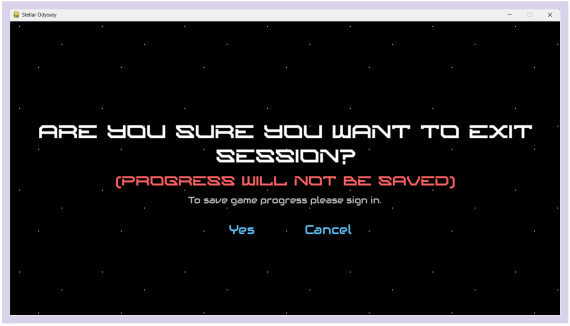


Zoom In: Zoomed in UI



Feature Fact Box: Fact box UI with factual information

Menu: In-game menu UI for settings and controls



Exit Session Message (Guest): Exist session message UI for signed-out players

→ 4.4 Version Control Summary

Sprint	Key Tasks/Commits
Main Game, Setup (Player Movement & Drawing Celestial Bodies)	Initialised game window, added player movement, implemented basic Solar System map
Home Menu (Play Button, Login Button & Help Button)	Created new separate screens/pages for Home Screen, Main Game, Login Page and Help Page
Fact Files	Implemented separate fact files unique to each celestial body, lines of facts and scrolling feature
Surfaces	Created surfaces as separate screens for players to explore
Fact Boxes	Implemented fact boxes contained with information on unique features of different celestial bodies
Collectables	Added collectable sprites to some surfaces
Inventory	Implemented inventory system to allow collection and viewing of collected items
User Authentication	Integrated user sign in/up system using encrypted storage
Save Progress	Established persistent save file for users with an account
UI Tips	Added several tips and clean UI for easy navigation, enhanced visual feedback
Sounds	Incorporated background music and sound effects
Testing	Refactored some code into classes, removed redundancy, tested code for bugs

---

5.0 Testing & Evaluation

→ 5.1 Testing Methods Used

- **Unit Test:** Testing individual modules (e.g. inventory system or user authentication)
- **Integration Test:** Testing how separate modules work together (e.g. login system + inventory persistence)
- **Test Data:** Inputs such as login credentials, planet names or key presses used to verify correct game behaviour
- **Dry Run:** Manual walkthrough of logic, using trace tables, to simulate code flow without execution

The tests I carried out were integrated in each development sprint. For example, after implementing the collectible logic, I unit tested the condition that checks if all fact boxes were viewed prior to showing the item. Integration tests ensured the item was properly saved and hidden in future sessions.

→ Justification of Methods:

What I Tested

- Authentication System
- Inventory System
- Surface Exploration Logic
- Fact Box Mechanics

Why These Methods Were Used

- Unit Testing was ideal after developing key modules like authentication or collectible logic to ensure intended performance after each sprint
- Integration Testing ensured smooth data flow between login, game state and inventory to ensure a streamlined user experience
- Dry Runs helped debug complex logic (e.g. conditional collectable appearance)
- Test Data allowed correct game behaviour to occur based on inputs

How This Suits My Project Goals

Stellar Odyssey consists of several features and interactions, so early and frequent testing ensured stability and prevented bugs from impacting later development. Keeping testing integrated with development helped iterate faster and confidently add new features.

Reflection

One key issue that occurred through unit testing was that collectibles still appeared after being collected. I resolved this by adding a conditional check to the rendering logic, preventing items from displaying if already in the player's save profile. This ensured consistent behaviour and improved user experience.

→ 5.2 Test Cases & Results

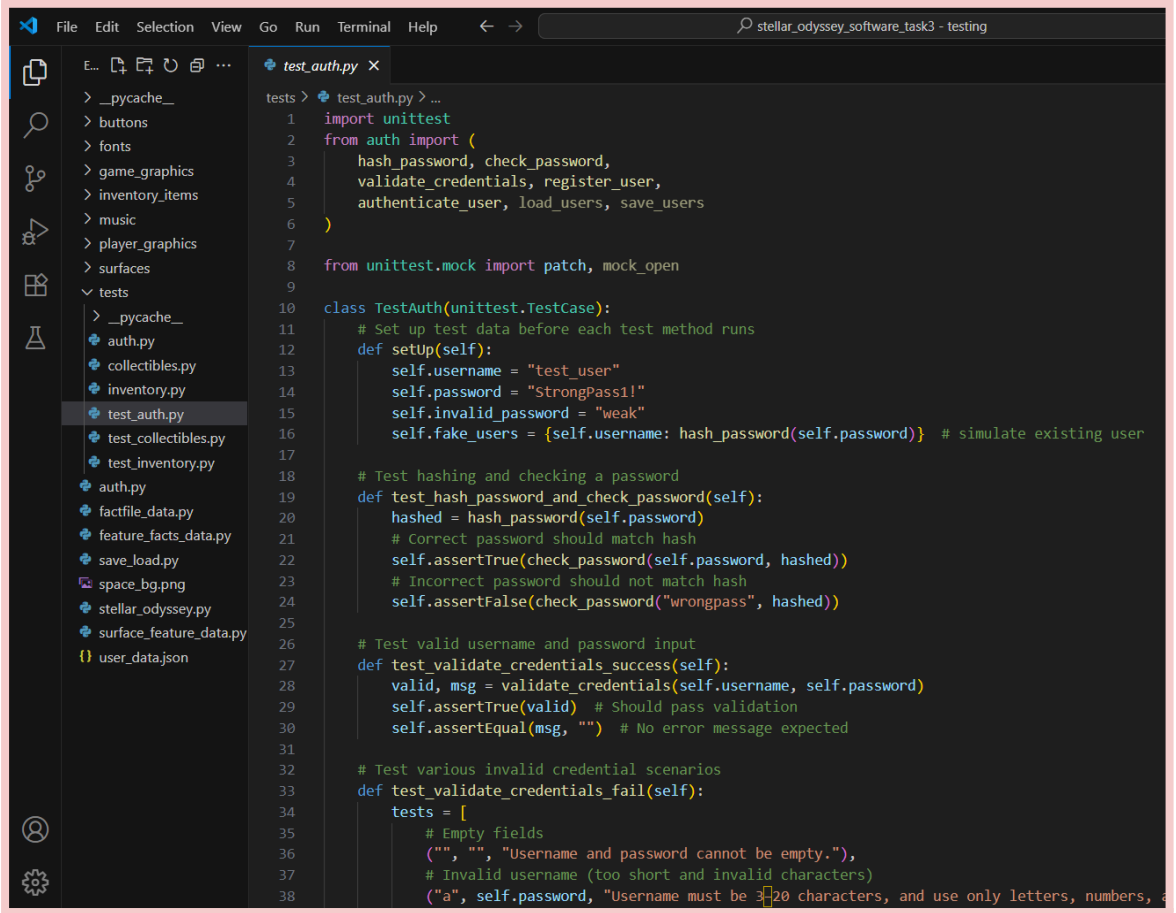
(1) Unit Test (Authentication, Collectible, Inventory)

Test ID	Description	Expected Result	Actual Result	Pass/Fail
UT01	Unit Test (Password Hashing)	Correct password matches hash (True), wrong one does not (False)	Matching works as expected	Pass
UT02	Unit Test (Credential Validation)	Valid credentials pass, invalid ones return specific error messages	Validation logic and messages are correct	Pass
UT03	Unit Test (User Registration)	Registration succeeds, and user is added to save lis	Returns success and user is stored	Pass
UT04	Unit Test (User Registration (Failure))	Registration fails with message: "Username already exists."	Returns failure with correct error	Pass
UT05	Unit Test (Authenticatio n Success)	Login succeeds (True)	Authenticatio n works and hash is validated correctly	Pass
UT06	Unit Test (Authenticatio n Failure)	Login fails (False)	Returns failure as user not in system	Pass
UT07	Unit Test (All Facts Open, Not Collected, Not in Inventory)	All facts are True, collected is False, and "Leaf" is not in the inventory	True	Pass
UT08	Unit Test	Even though	False	Pass

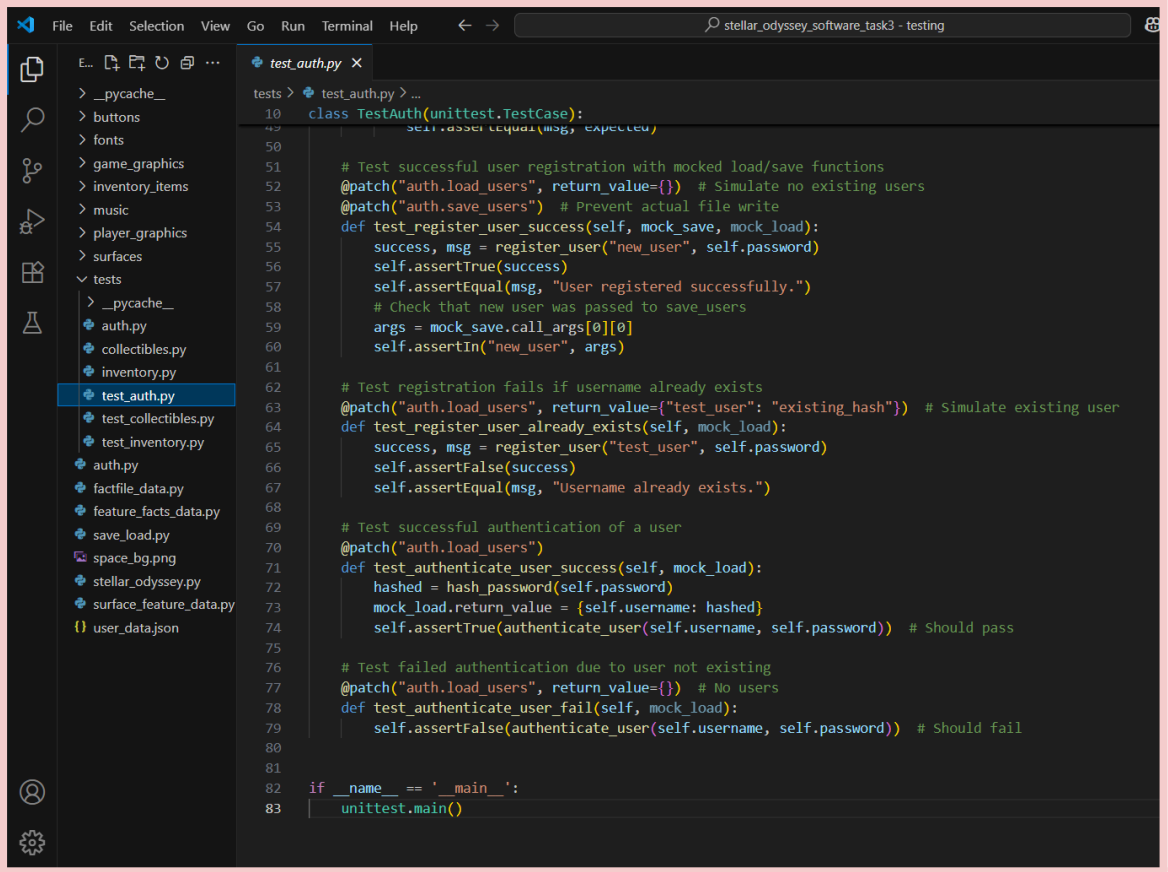
	(Already Collected)	all facts are True and inventory is empty, collected is True, so it should not be available		
UT09	Unit Test (Incomplete Facts)	One of the required facts ("b") is False, so the collectible shouldn't be available	False	Pass
UT10	Unit Test (Already in Inventory)	Even though the facts are True and collected is False, the item "Leaf" (already present as "leaf" in inventory) should prevent availability	False	Pass
UT11	Unit Test (Inventory Starts Empty)	[ ] (empty list)	[ ]	Pass
UT12	Unit Test (Add Single Item)	Item "Plasma" appears in inventory	Item "Plasma" found in inventory	Pass
UT13	Unit Test (Prevent Duplicate Items)	Inventory contains only one instance of "Plasma"	Inventory length is 1	Pass
UT14	Unit Test (Remove Existing Ite	Item "Leaf" no longer present	Item "Leaf" not found in inventory	Pass
UT15	Unit Test (Ignore Missing Item on Remove)	Inventory remains unchanged (still empty)	Inventory is still empty	Pass

UT16	Unit Test (Check Item Presence)	Returns True for "Blueberries", False for "Rock"	Returned True for "Blueberries", False for "Rock"	Pass
UT17	Unit Test (Ensure Inventory Copy Integrity)	"Fake" not added to actual inventory	"Fake" not found in inventory	Pass

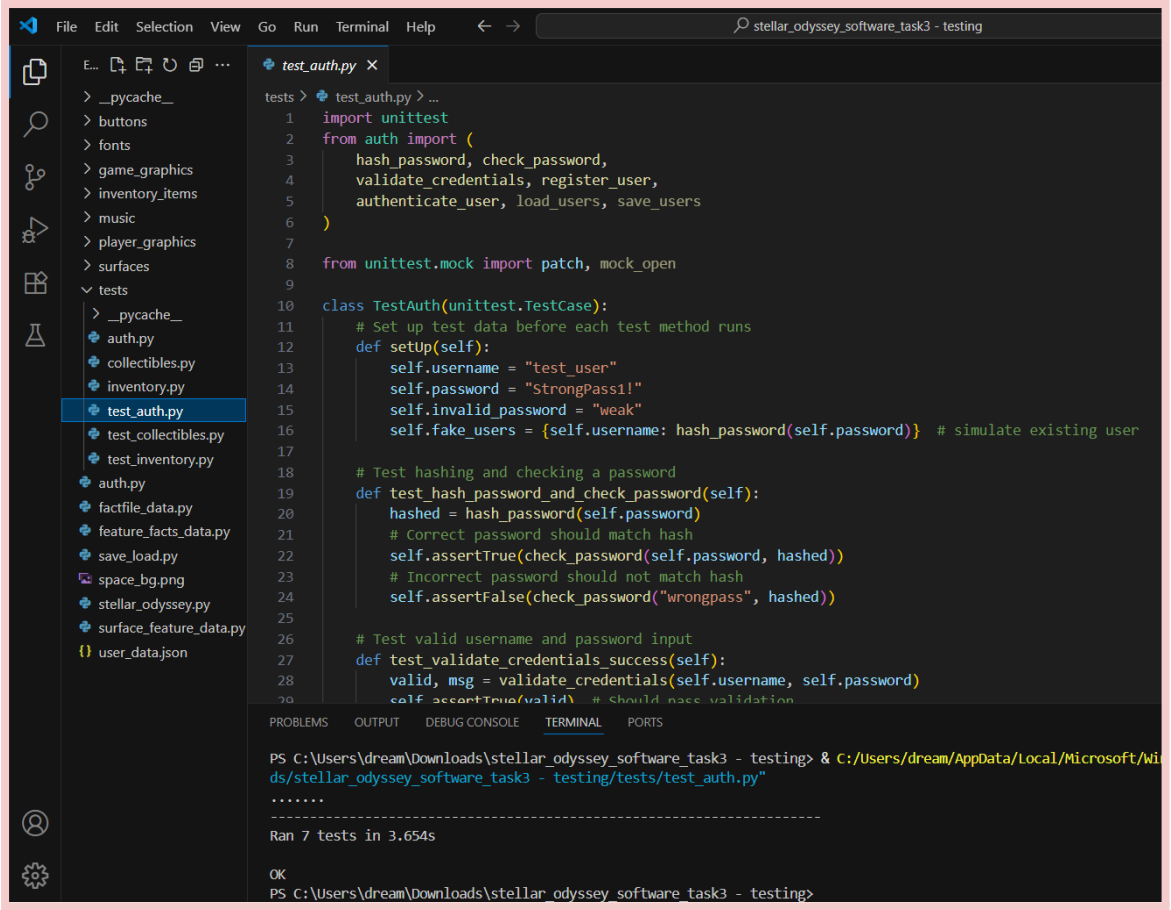
Unit Test (Script Examples)



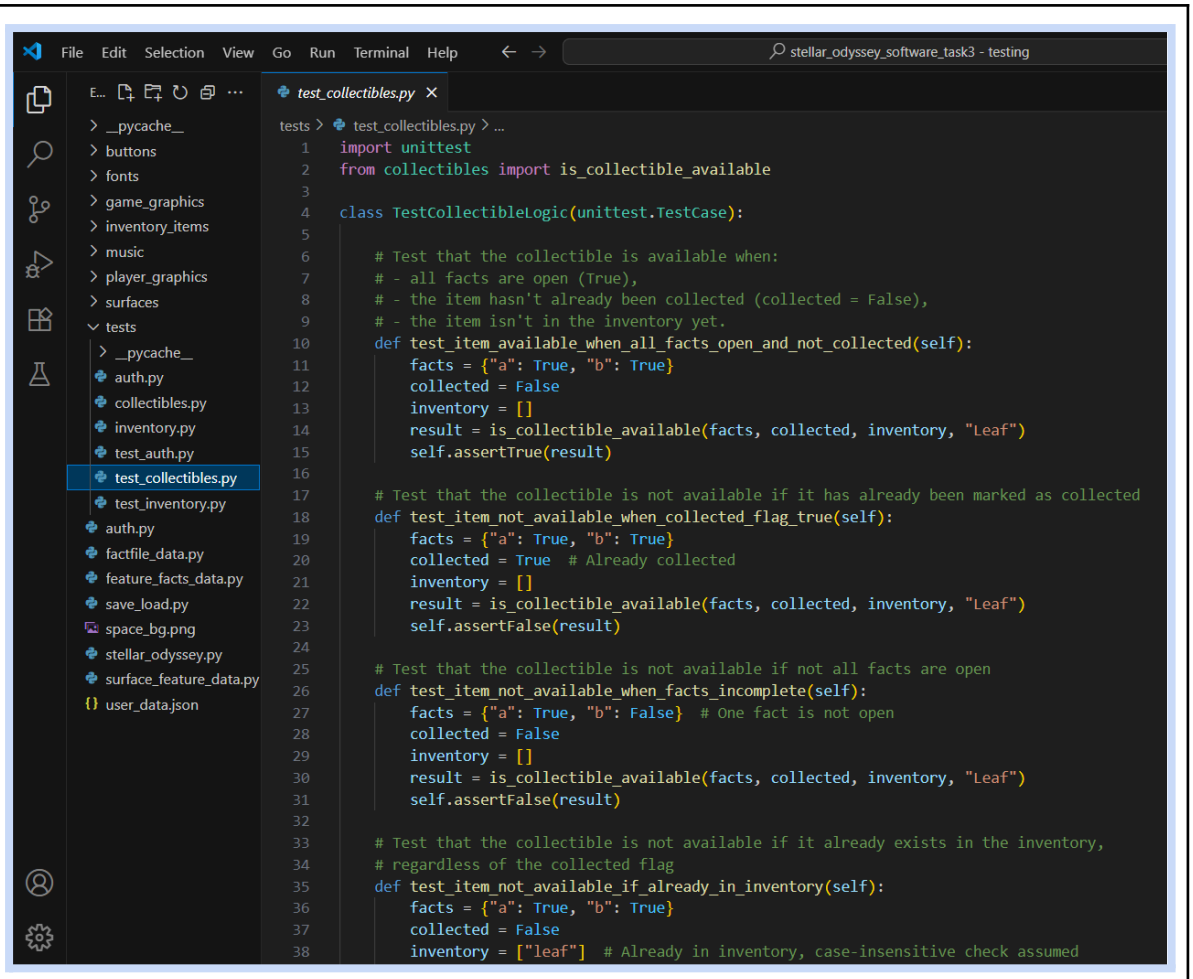
Unit Test Code (Authentication Script)



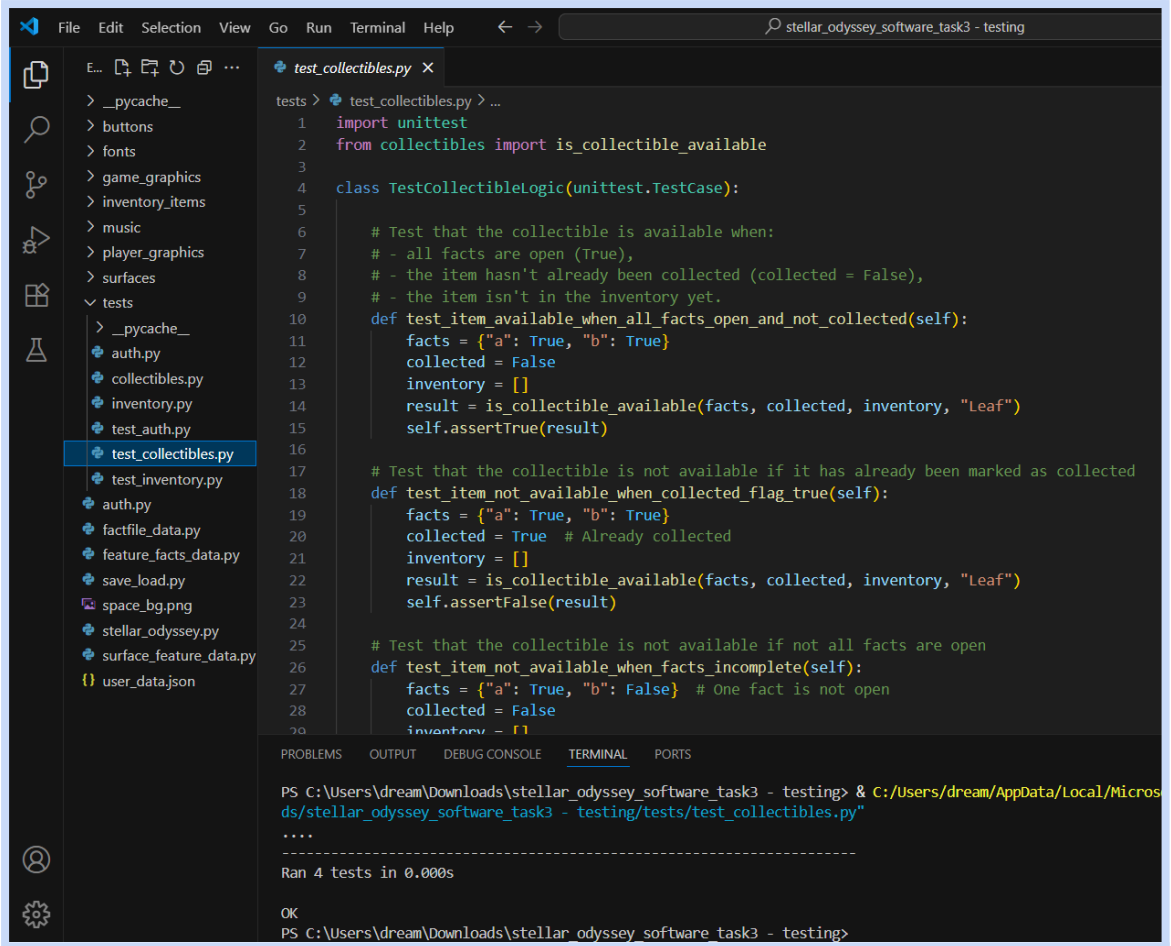
Unit Test Code (Authentication Script) #2



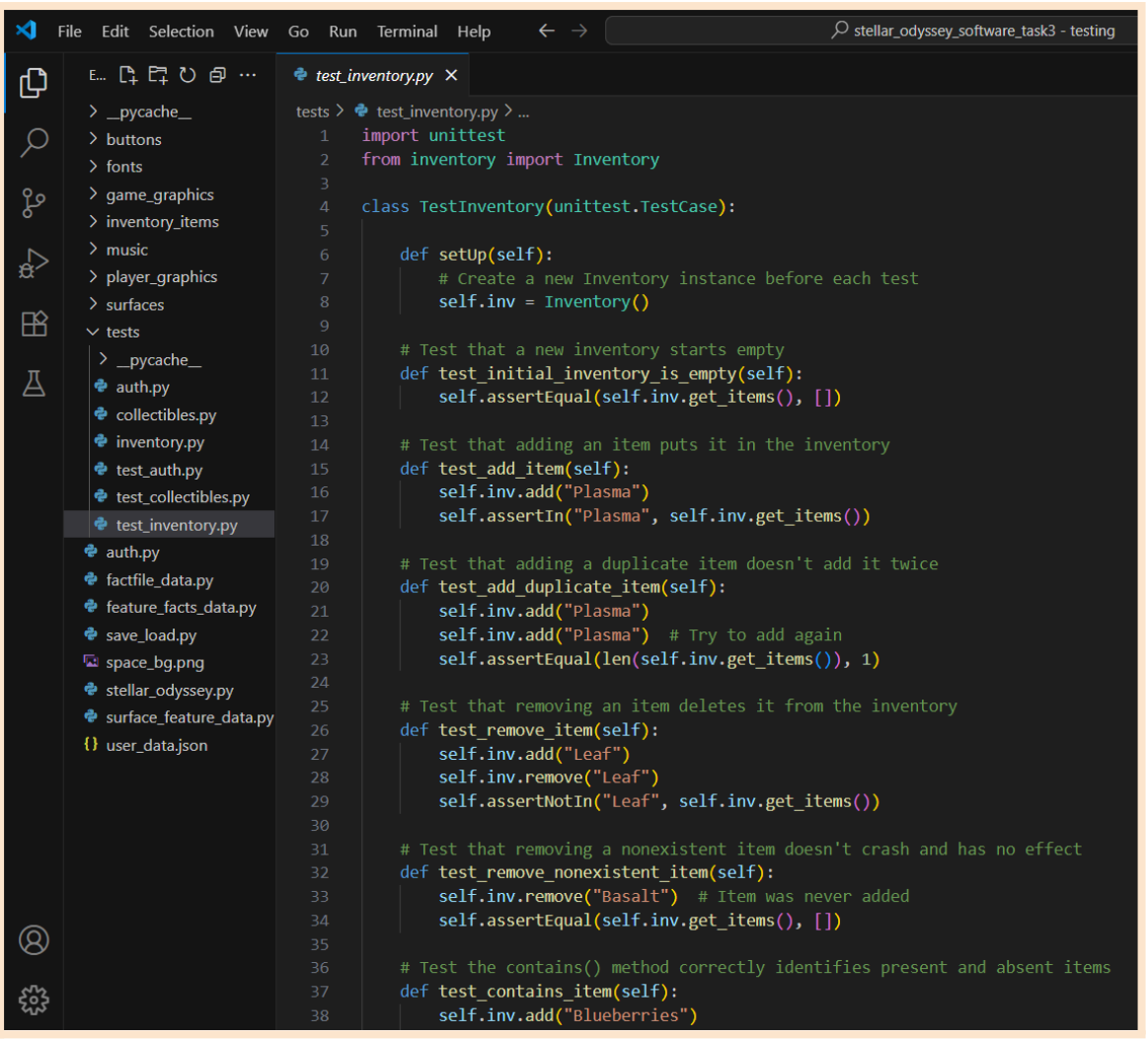
Unit Test Code (Authentication Script) - Result



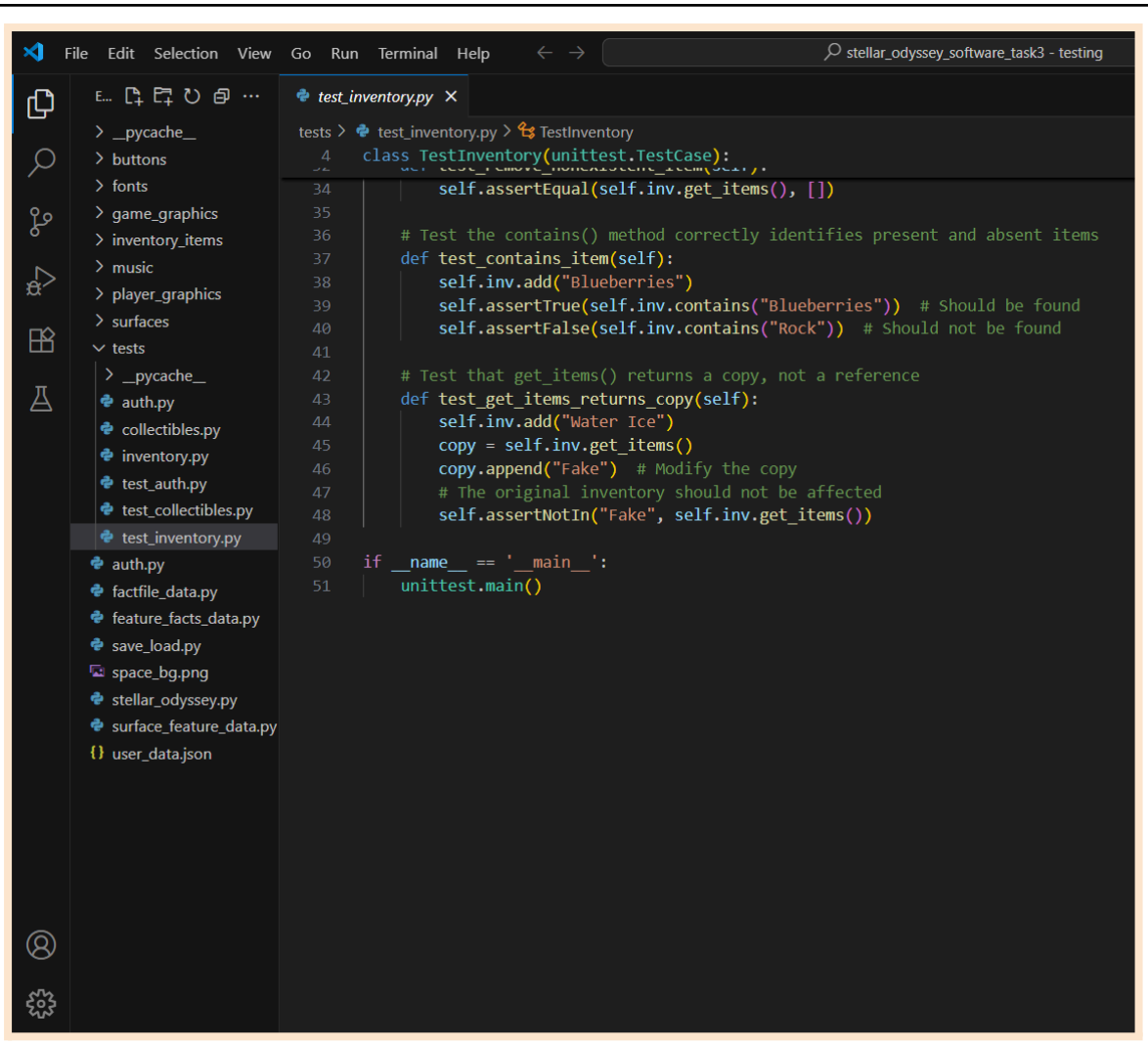
Unit Test Code (Collectibles Script)



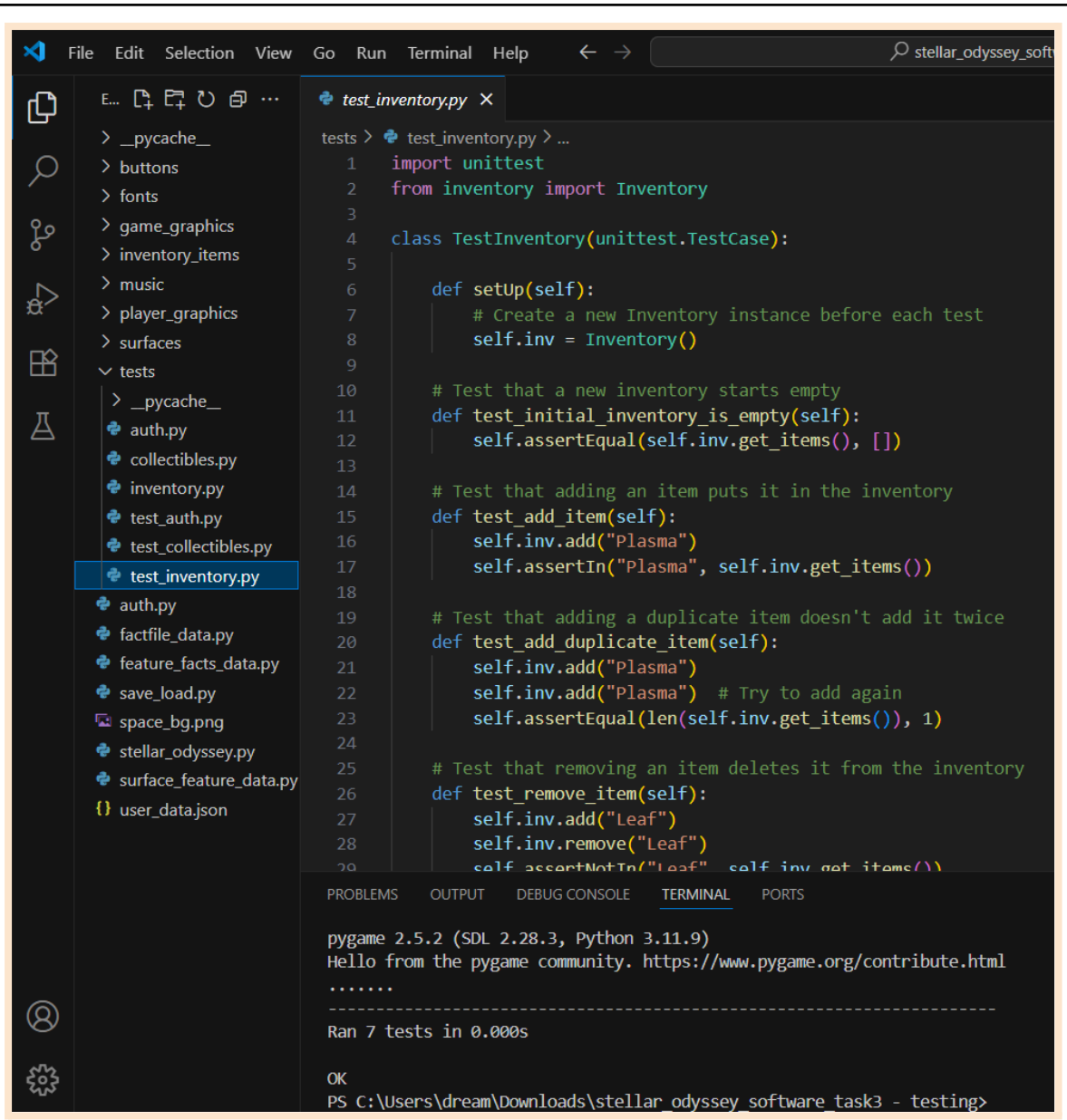
Unit Test Code (Collectibles Script) - Result



Unit Test Code (Inventory Script)



Unit Test Code (Inventory Script) #2



Unit Test Code (Inventory Script) - Result

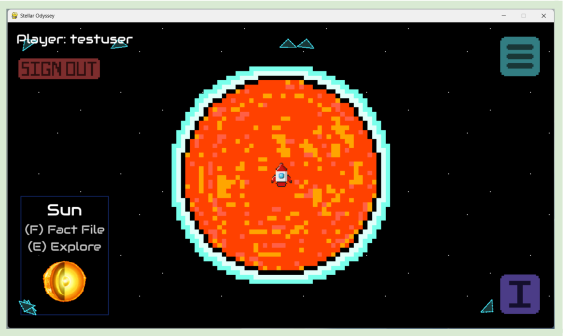
(2) Integration Test (Login + Inventory Persistence + Collectible Logic)

Test ID	Description	Expected Result	Actual Result	Pass/Fail
IT01	Login	User logs in successfully	Logged in to testuser	Pass
IT02	Enter Surface	Surface loads with fact boxes	Mars surface loaded correctly	Pass
IT03	View All Facts	Collectible	Blueberries	Pass

		becomes visible	appeared	
IT04	Collect Item	Item added to inventory	Collection successful	Pass
IT05	Quit Game	Game exits safely	Exited with no errors	Pass
IT06	Relaunch/Log in	Same user logs in successfully	testuser login worked	Pass
IT07	Inventory Load	Previously collected item is still visible	Blueberries present	Pass
IT08	Revisist Surface	Collectible does not reappear	No Blueberries on surface	Pass

Integration Test (In-Game Examples)

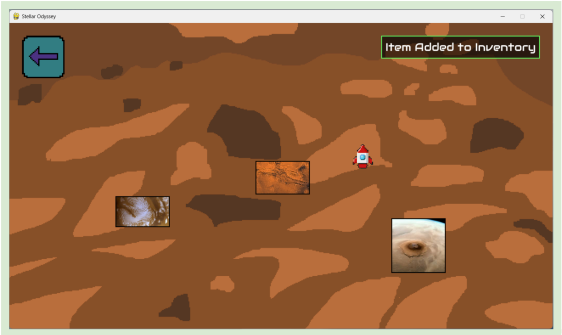
‘testuser’ player credentials created in sign up page successfully



‘testuser’ player profile successfully appears (top left)

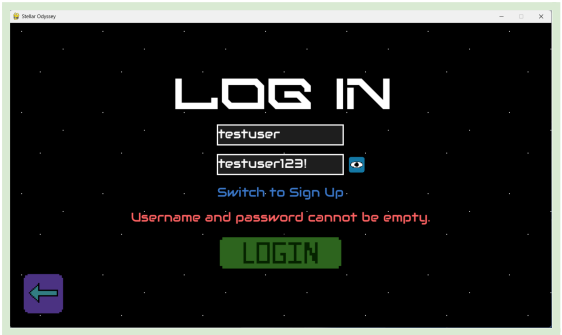
‘testuser’ player enters Mars with correctly drawn sprites and text

‘testuser’ player opens all fact boxes and collectable item appears



‘testuser’ player collects item and is added to inventory successfully

‘testuser’ player closes the game with no errors



‘testuser’ player logs back in to the game successfully

‘testuser’ player opens inventory and finds item saved successfully

No collectable item appears as ‘testuser’ player already collected it

Overall all main components of Login, Inventory Persistence and Collectible Logic all work together properly and successfully.

(3) Test Data (Input Validation & Response)

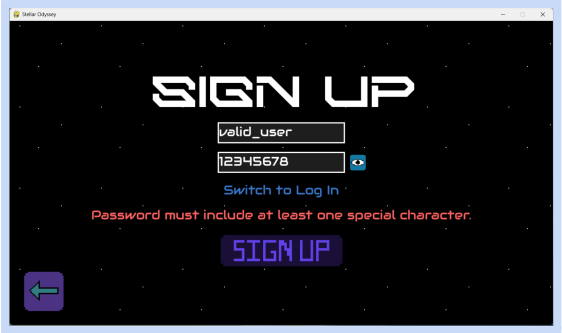
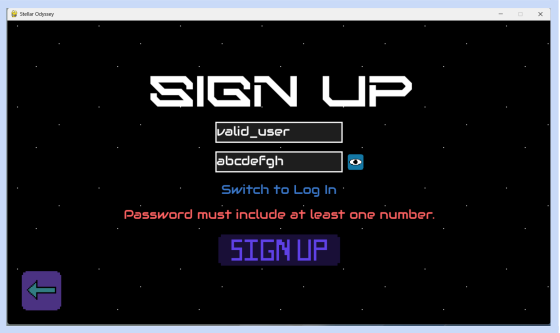
Test ID	Input/Test Data	Expected Result	Actual Result	Pass/Fail
TD01	!nv@lid_user	Registration rejected due to invalid characters	Error message shown, registration blocked	Pass

TD02	1	Password rejected as too short	Error message shown	Pass
TD03	12345678	Password rejected as no special characters	Error message shown, registration blocked	Pass
TD04	abcdefg	Password rejected as no numbers	Error message shown, registration blocked	Pass
TD05	Player / player123!	Registration succeeds	Account created, game started	Pass
TD06	'Z' during Solar System view	No action, game continues normally	No response, no issues	Pass
TD07	WASD	Player moves correctly	Smooth movement observed	Pass
TD08	'E' when touching celestial body	Surface screen loads	Transition to surface worked	Pass
TD09	'C' with no item nearby	No collection, no error	No change, as expected	Pass

Test Data (In-Game Examples)

Test ID: TD01

Test ID: TD02

	
Test ID: TD03	**Test ID: TD04**

→ 5.3 Evaluation Against Requirements

My game met both functional and non-functional requirements effectively through:

- **Player Movement:** Smooth and responsive across all intended screens
- **Collision Detection & Interaction:** Accurate triggering of fact boxes, collectible pickup and surface entry
- **Separate Display Screens:** Menu, Solar System and other surface views work independently with seamless transitions
- **Data Storage:** User progress is securely saved and loaded correctly using a JSON-based login and inventory system
- **Input Detection:** Keyboard and mouse inputs are consistently detected and handled

Non-functional requirements including performance (consistent 60 fps), user-friendly interface, consistent aesthetics, portability and basic sound effects were also successfully implemented, contributing to a positive and accessible user experience.

→ 5.4 Improvements & Future Work

- **Code Refactoring:** Organise the codebase more effectively by refactoring larger sections into separate classes, particularly for the player and game state
- **Inventory Enhancements:** Add fact boxes for each collected items to provide further educational value
- **Surface Expansion:** Extend surfaces with scrolling features to enable deeper exploration
- **Additional Features:** Overall include more surface details, mmin-game or dynamic environmental effects to further engage players

---

6.0 (Client) Feedback & Reflection

→ 6.1 Summary of Client Feedback

Peer	Feedback
Andrew Lou	Overall the game is well thought out and is smooth and accessible by everyone, though some information is wrong as it is not to scale (Jupiter).
Joshua Eum	There are some aspects that can be improved. In particular, the sizes of the planets could be made more accurate (Jupiter). One more thing to note is the lack of diagonal movement animation in the player, making the movement feel kind of blocky. The implementation of items is a smart move, but I think the system would benefit by adding item descriptions. Overall this is a good game.
Daniel Hay	This application has creative core features that combine to create a unique concept that provides educational content in a fun manner.

→ 6.2 Personal Reflection

Throughout this project, I developed confident skills in managing a full software development process, from planning to implementation. I deepened my understanding of Python and the Pygame library, especially in building interactive systems such as player movement, UI handling, surface transitions and local data storage. Key skills I gained include:

- **Programing:** Implementing gameplay mechanics, handling user input and managing game states using Python and Pygame
- **Data Management:** Using JSON files for saving progress and applying encryption with bcrypt for secure login systems
- **UI/UX Design:** Creating clean and intuitive interfaces suitable for users of all ages
- **Project Management:** Breaking down development into Agile-style sprints, managing time effectively and create features incrementally
- **Problem Solving:** Debugging issues, running tests and acting on feedback helped improve both code quality and gameplay experience

Ultimately, this project helped me understand the importance of iterative development, structured planning and user-centred design in creating a functional and engaging educational game.

7.0 Appendices

- **Full Gantt Chart**
  → 2.3
- **Complete Data Dictionary**
  → 3.5
- **Full Test Logs**
  → 5.2
- **Raw Interview Notes / Feedback Forms**
  → 6.1
- **Exemplar Code Snippets**
  → 5.2 (Unit Tests)
  → Classes (Below)

```python
"""Stellar Odyssey - Classes"""

# SurfaceFeatureSet Class
class SurfaceFeatureSet:
    # Initialise SurfaceFeatureSet with list of features
    def __init__(self, features):
        self.features = []
        for feature in features: # Load each feature's normal and touch images
            normal_img = pygame.image.load(f"game_graphics/surface_graphics/{feature['name']}.png").convert_alpha()
            touch_img = pygame.image.load(f"game_graphics/surface_graphics/{feature['name']}_touch.png").convert_alpha()
            rect = normal_img.get_rect(topleft=feature["pos"])
            self.features.append({
                "name": feature["name"],
                "image": normal_img,
                "touch_image": touch_img,
                "rect": rect,
                "hovered": False
            })
        self.current_hovered_feature = None

    # Draw all features on screen, highlighting hovered feature
    def draw(self, screen, player_rect, mouse_pos):
        self.current_hovered_feature = None
        for feature in self.features: # Iterate through each feature
            is_touching = player_rect.colliderect(feature["rect"])
            is_hovering = feature["rect"].collidepoint(mouse_pos)
            feature["hovered"] = is_touching or is_hovering

            current_img = feature["touch_image"] if feature["hovered"] else feature["image"]
            screen.blit(current_img, feature["rect"].topleft)

            if feature["hovered"]: # If feature is hovered or touched, display its label
                self.current_hovered_feature = feature
                display_name = feature["name"].replace("_", " ").title()
                label = surface_label_font.render(display_name, True, (255, 255, 255))
                label_rect = label.get_rect(midbottom=(feature["rect"].centerx, feature["rect"].top - 10))
```

```python
class SurfaceFeatureSet:
    def draw(self, screen, player_rect, mouse_pos):

                padding = 8
                bg_rect = pygame.Rect(
                    label_rect.left - padding,
                    label_rect.top - padding,
                    label_rect.width + 2 * padding,
                    label_rect.height + 2 * padding
                )
                bg_surface = pygame.Surface((bg_rect.width, bg_rect.height), pygame.SRCALPHA)
                bg_surface.fill((0, 0, 0, 160))
                screen.blit(bg_surface, (bg_rect.left, bg_rect.top))

                screen.blit(label, label_rect)

    # Handle mouse events for features
    def get_hovered_feature(self):
        return self.current_hovered_feature
```

```python
# InputBox Class
class InputBox:
    # Initialise InputBox with position, size, font and password mode
    def __init__(self, x, y, w, h, font, is_password=False):
        self.rect = pygame.Rect(x, y, w, h)
        self.font = font
        self.color_inactive = pygame.Color('white')
        self.color_active = pygame.Color('deepskyblue')
        self.color_hover = pygame.Color('lightskyblue')
        self.color = self.color_inactive
        self.text = ''
        self.txt_surface = font.render('', True, self.color)
        self.active = False
        self.is_password = is_password
        self.show_password = False
        self.scroll_offset = 0

    # Handle events for the input box
    def handle_event(self, event):
        if event.type == pygame.MOUSEBUTTONDOWN:
            self.active = self.rect.collidepoint(event.pos) # Check if clicked inside the box
        if event.type == pygame.KEYDOWN and self.active:
            if event.key == pygame.K_RETURN: # Continue when 'Enter' key pressed
                return 'submit'
            elif event.key == pygame.K_BACKSPACE: # Remove character when backspace pressed
                self.text = self.text[:-1]
            else: # Handle other keys
                if len(self.text) < 20:
                    self.text += event.unicode
            if self.is_password: # If password mode, update display text
                display_text = self.text if self.show_password else '*' * len(self.text)
            else:
                display_text = self.text
            self.txt_surface = self.font.render(display_text, True, self.color)

            text_width = self.txt_surface.get_width()
```

```
252  class InputBox:
269      def handle_event(self, event):
286              text_width = self.txt_surface.get_width()
287              if text_width > self.rect.width - 20: # If text exceeds box width, enable scrolling
288                  self.scroll_offset = text_width - (self.rect.width - 20)
289              else:
290                  self.scroll_offset = 0
291
292          return None
293
294      # Update input box state
295      def draw(self, screen):
296          mouse_pos = pygame.mouse.get_pos()
297          if self.rect.collidepoint(mouse_pos): # Check if mouse is hovering over box
298              if not self.active: # If box not active, change colour to hover colour
299                  border_color = self.color_hover
300              else:
301                  border_color = self.color_active
302          else:
303              border_color = self.color_active if self.active else self.color_inactive
304
305          pygame.draw.rect(screen, (30, 30, 30), self.rect)
306          pygame.draw.rect(screen, border_color, self.rect, 3)
307
308          # Create surface for text and fill with a dark colour
309          text_clip_surface = pygame.Surface((self.rect.width - 6, self.rect.height - 6))
310          text_clip_surface.fill((30, 30, 30))
311
312          text_clip_surface.blit(self.txt_surface, (-self.scroll_offset, 0))
313
314          screen.blit(text_clip_surface, (self.rect.x + 3, self.rect.y + 3))
315
316      # Get current text in input box
317      def get_text(self):
318          return self.text
```

```
252  class InputBox:
319
320      # Clear input box text
321      def clear(self):
322          self.text = ''
323          self.txt_surface = self.font.render('', True, self.color)
324
325      # Toggle password visibility
326      def toggle_password_visibility(self):
327          if self.is_password: # Only toggle if in password mode
328              self.show_password = not self.show_password
329              display_text = self.text if self.show_password else '*' * len(self.text)
330              self.txt_surface = self.font.render(display_text, True, self.color)
```

```python
# FactFile Class
class FactFile:
    # Initialise FactFile with planet data
    def __init__(self, name, classification, size, mass, distance_from_sun, temperature,
                 position, gravity, rotation_period, orbital_period, composition=None):
        self.name = name
        self.classification = classification
        self.size = size
        self.mass = mass
        self.distance_from_sun = distance_from_sun
        self.temperature = temperature
        self.position = position
        self.gravity = gravity
        self.rotation_period = rotation_period
        self.orbital_period = orbital_period
        self.scroll_offset = 0
        self.scroll_velocity = 0
        self.scroll_speed = 600
        self.scroll_direction = 0
        self.dragging_scrollbar = False
        self.drag_start_y = 0
        self.initial_scroll_offset = 0
        self.composition = composition or {}

    # Handle scroll events for fact file
    def handle_scroll(self, event):
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP: # Scroll up when 'Up' key pressed
                self.scroll_direction = 1
            elif event.key == pygame.K_DOWN: # Scroll down when 'Down' key pressed
                self.scroll_direction = -1
        elif event.type == pygame.KEYUP: # Stop scrolling when key released
            if event.key in (pygame.K_UP, pygame.K_DOWN):
                self.scroll_direction = 0

```

```
333  class FactFile:
357      def handle_scroll(self, event):
365                  self.scroll_direction = 0
366
367          elif event.type == pygame.MOUSEBUTTONDOWN:
368              if event.button == 4: # Scroll up with mouse wheel
369                  self.scroll_offset += 60
370              elif event.button == 5: # Scroll down with mouse wheel
371                  self.scroll_offset -= 60
372              elif event.button == 1: # Left mouse button pressed
373                  mouse_x, mouse_y = event.pos
374                  thumb_rect = self._get_scroll_thumb_rect()
375                  if thumb_rect.collidepoint(mouse_x, mouse_y): # Check if clicked on scrollbar thumb
376                      self.dragging_scrollbar = True
377                      self.drag_start_y = mouse_y
378                      self.initial_scroll_offset = self.scroll_offset
379
380          elif event.type == pygame.MOUSEBUTTONUP: # Release mouse button
381              if event.button == 1:
382                  self.dragging_scrollbar = False
383
384          elif event.type == pygame.MOUSEMOTION: # Mouse moved
385              if self.dragging_scrollbar: # If dragging scrollbar
386                  dy = event.pos[1] - self.drag_start_y
387                  content_height = self._get_content_height()
388                  visible_height = HEIGHT // 2
389                  max_offset = max(1, content_height - visible_height)
390                  scrollable_area = visible_height - self._get_thumb_height(content_height)
391                  if scrollable_area > 0: # Prevent division by zero
392                      scroll_ratio = dy / scrollable_area
393                      self.scroll_offset = self.initial_scroll_offset - scroll_ratio * max_offset
394
395      # Update scroll offset based on scroll speed and direction
396      def update_scroll(self, dt):
397          self.scroll_velocity = self.scroll_speed * self.scroll_direction
398          self.scroll_offset += self.scroll_velocity * dt
399          self.scroll_offset = min(0, self.scroll_offset)
```

```
333  class FactFile:
401      # Draw fact file on screen
402      def draw(self, screen):
403          draw_parallax_bg()
404
405          # Set Up
406          name_font = pygame.font.Font("fonts/stellar.otf", 80)
407          content_font = default_font
408          composition_font = pygame.font.Font("fonts/audiowide.ttf", 36)
409
410          # Render Fact File Content
411          lines = [
412              f"{self.name}",
413              f"Classification: {self.classification}",
414              f"Position in Solar System: {self.position}",
415              f"Distance from Sun: {self.distance_from_sun}",
416              f"Diameter: {self.size}",
417              f"Mass: {self.mass}",
418              f"Surface Gravity: {self.gravity}",
419              f"Rotation Period: {self.rotation_period}",
420              f"Orbital Period: {self.orbital_period}",
421              f"Average Surface Temperature: {self.temperature}"
422          ]
423
424          title_text = factfile_name_font.render(lines[0], True, (255, 255, 255))
425
426          line_spacing = 20
427          start_y = HEIGHT // 4 + self.scroll_offset
428
429          for line in lines[1:]: # Render each line of fact file
430              text_surface = factfile_content_font.render(line, True, (255, 255, 255))
431              faded_surface = factfile_content_font.render(line, True, (180, 180, 180))
432
433          content_height = self._get_content_height()
434          visible_height = HEIGHT // 2
435          min_scroll = min(0, visible_height - content_height)
```

```
333  class FactFile:
402      def draw(self, screen):
436
437          if self.scroll_offset < min_scroll: # Clamp scroll offset to minimum
438              self.scroll_offset = min_scroll
439          elif self.scroll_offset > 0: # Clamp scroll offset to maximum
440              self.scroll_offset = 0
441
442          # Draw Title
443          title_y = 50 + self.scroll_offset
444          title_text = name_font.render(lines[0], True, (255, 255, 255))
445          screen.blit(title_text, (solar_system_button_rect.right + 20, title_y))
446
447          # Draw Divider
448          divider_y = title_y + name_font.get_height() + 10
449          pygame.draw.line(screen, (100, 100, 100), (solar_system_button_rect.right + 20, divider_y), (WIDTH - 60, divider_y), 2)
450
451          y_offset = divider_y + 30
452
453          mouse_x, mouse_y = pygame.mouse.get_pos()
454
455          for line in lines[1:]: # Render each line of fact file
456              wrapped_lines = wrap_text(line, content_font, WIDTH - 100)
457              for wrapped_line in wrapped_lines: # Wrap text to fit within width
458                  text_surface = content_font.render(wrapped_line, True, (255, 255, 255))
459                  faded_surface = content_font.render(wrapped_line, True, (180, 180, 180))
460
461                  text_rect = text_surface.get_rect(topleft=(solar_system_button_rect.right + 20, y_offset))
462
463                  if text_rect.collidepoint(mouse_x, mouse_y): # Check if mouse is hovering over text
464                      screen.blit(text_surface, text_rect.topleft)
465                  else:
466                      screen.blit(faded_surface, text_rect.topleft)
467
468                  y_offset += content_font.get_height() + 10
469
470              y_offset += 60
```

```
333  class FactFile:
402      def draw(self, screen):
470              y_offset += 60
471
472          y_offset += 10
473          pygame.draw.line(screen, (150, 150, 150), (solar_system_button_rect.right + 20, y_offset),
474                           (WIDTH - 60, y_offset), 2)
475          y_offset += 30
476
477          composition_title = composition_font.render("Composition", True, (255, 255, 255))
478          screen.blit(composition_title, (solar_system_button_rect.right + 20, y_offset))
479          y_offset += composition_title.get_height() + 20
480
481          for layer, details in self.composition.items(): # Render composition details
482              layer_title = content_font.render(f"{layer}:", True, (200, 200, 200))
483              screen.blit(layer_title, (solar_system_button_rect.right + 20, y_offset))
484              y_offset += layer_title.get_height() + 10
485
486              wrapped_lines = wrap_text(details, content_font, WIDTH - 250)
487              for line in wrapped_lines: # Wrap text to fit within width
488                  text_surface = content_font.render(line, True, (255, 255, 255))
489                  faded_surface = content_font.render(line, True, (180, 180, 180))
490                  text_rect = text_surface.get_rect(topleft=(solar_system_button_rect.right + 40, y_offset))
491
492                  if text_rect.collidepoint(mouse_x, mouse_y): # Check if mouse is hovering over text
493                      screen.blit(text_surface, text_rect.topleft)
494                  else:
495                      screen.blit(faded_surface, text_rect.topleft)
496
497                  y_offset += content_font.get_height() + 20
498
499              y_offset += 20
500
501          self._composition_height = y_offset
502
503          # Draw Scrollbar
504          scrollbar_height = HEIGHT // 2
```

```
333  class FactFile:
402      def draw(self, screen):
504          scrollbar_height = HEIGHT // 2
505          scrollbar_y = HEIGHT // 4
506          pygame.draw.rect(screen, (50, 50, 50), (WIDTH - 40, scrollbar_y, 20, scrollbar_height))
507
508          thumb_rect = self._get_scroll_thumb_rect()
509          mouse_x, mouse_y = pygame.mouse.get_pos()
510          hovered = thumb_rect.collidepoint(mouse_x, mouse_y)
511          thumb_color = (220, 220, 220, 255) if hovered else (180, 180, 180, 180)
512          thumb_surface = pygame.Surface((thumb_rect.width, thumb_rect.height), pygame.SRCALPHA)
513          thumb_surface.fill(thumb_color)
514          screen.blit(thumb_surface, thumb_rect.topleft)
515
516          screen.blit(solar_system_button, solar_system_button_rect.topleft)
517
518      # Calculate total height of content for scrolling
519      def _get_content_height(self):
520          base_line_spacing = 60
521          base_lines = 10
522          composition_lines = 1
523
524          for layer, details in self.composition.items(): # Calculate height of composition details
525              composition_lines += 1
526              wrapped = wrap_text(details, factfile_content_font, WIDTH - 100)
527              composition_lines += len(wrapped)
528
529          return 80 + base_lines * (30 + base_line_spacing) + composition_lines * (factfile_content_font.get_height() + 25) + 20 * len(self.composit
530
531      # Calculate height of scrollbar thumb based on content height
532      def _get_thumb_height(self, content_height):
533          scrollbar_height = HEIGHT // 2
534          return max(scrollbar_height * scrollbar_height // content_height, 30)
535
536      # Get rectangle for scrollbar thumb based on scroll offset
537      def _get_scroll_thumb_rect(self):
538          content_height = self._get_content_height()
```


```
333  class FactFile:
402      def draw(self, screen):
504          scrollbar_height = HEIGHT // 2
505          scrollbar_y = HEIGHT // 4
506          pygame.draw.rect(screen, (50, 50, 50), (WIDTH - 40, scrollbar_y, 20, scrollbar_height))
507
508          thumb_rect = self._get_scroll_thumb_rect()
509          mouse_x, mouse_y = pygame.mouse.get_pos()
510          hovered = thumb_rect.collidepoint(mouse_x, mouse_y)
511          thumb_color = (220, 220, 220, 255) if hovered else (180, 180, 180, 180)
512          thumb_surface = pygame.Surface((thumb_rect.width, thumb_rect.height), pygame.SRCALPHA)
513          thumb_surface.fill(thumb_color)
514          screen.blit(thumb_surface, thumb_rect.topleft)
515
516          screen.blit(solar_system_button, solar_system_button_rect.topleft)
517
518      # Calculate total height of content for scrolling
519      def _get_content_height(self):
520          base_line_spacing = 60
521          base_lines = 10
522          composition_lines = 1
523
524          for layer, details in self.composition.items(): # Calculate height of composition details
525              composition_lines += 1
526              wrapped = wrap_text(details, factfile_content_font, WIDTH - 100)
527              composition_lines += len(wrapped)
528
529          return 80 + base_lines * (30 + base_line_spacing) + composition_lines * (factfile_content_font.get_height() + 25) + 20 * len(self.composit
530
531      # Calculate height of scrollbar thumb based on content height
532      def _get_thumb_height(self, content_height):
533          scrollbar_height = HEIGHT // 2
534          return max(scrollbar_height * scrollbar_height // content_height, 30)
535
536      # Get rectangle for scrollbar thumb based on scroll offset
537      def _get_scroll_thumb_rect(self):
538          content_height = self._get_content_height()
```

```python
# Planet Class
class Planet:
    # Initialise Planet with name, image path, size, radius, speed and angle
    def __init__(self, name, image_path, size, radius, speed, angle=None):
        self.name = name
        self.original_image_path = image_path
        self.touch_image_path = image_path.replace(".png", "_touch.png")
        self.size = size
        self.radius = radius
        self.speed = speed
        self.angle = angle if angle is not None else random.uniform(0, 2 * math.pi)
        self.image = pygame.transform.scale(pygame.image.load(image_path).convert_alpha(), size)
        self.original_image = self.image.copy()
        self.touched = False
        self.position = (0, 0)

    # Update planet position based on angle and speed
    def update_position(self, center, dt):
        self.angle += self.speed * dt
        cx, cy = center
        self.position = (
            cx + math.cos(self.angle) * self.radius - self.size[0] // 2,
            cy + math.sin(self.angle) * self.radius - self.size[1] // 2
        )

    # Update touch state based on player rectangle
    def update_touch_state(self, player_rect):
        planet_rect = pygame.Rect(self.position[0], self.position[1], self.size[0], self.size[1])
        if planet_rect.colliderect(player_rect): # Check if player touching planet
            if not self.touched: # If not already touched, change image to touch image
                try:
                    self.image = pygame.transform.scale(pygame.image.load(self.touch_image_path).convert_alpha(), self.size)
                except:
                    pass
                self.touched = True
        else: # If player not touching planet, reset image to original
            if self.touched:
```

```python
class Planet:
    def update_touch_state(self, player_rect):
            if self.touched:
                self.image = self.original_image.copy()
                self.touched = False

    # Draw planet on screen at its current position
    def draw(self, screen, camera_x, camera_y, zoom):
        scaled = pygame.transform.smoothscale(self.image, (int(self.size[0] * zoom), int(self.size[1] * zoom)))
        screen.blit(scaled, ((self.position[0] - camera_x) * zoom, (self.position[1] - camera_y) * zoom))
```

```python
class Collectible:
    def __init__(self, name, planet, fact_flags, collected_attr, img, touch_img, rect, label_color):
        self.name = name
        self.planet = planet
        self.fact_flags = fact_flags
        self.collected_attr = collected_attr
        self.img = img
        self.touch_img = touch_img
        self.rect = rect
        self.label_color = label_color

    # Check if collectible is available to collect
    def is_available(self):
        if all(self.fact_flags.values()) and not getattr(sys.modules[__name__], self.collected_attr): # Check if all fact flags are true and coll
            normalized_inventory = [item.lower().replace(" ", "_") for item in player_inventory.get_items()]
            normalized_name = self.name.lower().replace(" ", "_")
            return normalized_name not in normalized_inventory

        return False

    # Draw collectible on screen
    def draw(self, screen, player_rect, label_font):
        if not self.is_available(): # If collectible not available, do not draw
            return

        hovered = player_rect.colliderect(self.rect)
        screen.blit(self.touch_img if hovered else self.img, self.rect.topleft)

        if hovered: # If collectible is hovered, display its label
            label = label_font.render(self.name, True, (255, 255, 255))
            label_rect = label.get_rect(midbottom=(self.rect.centerx, self.rect.top - 10))
            bg_rect = pygame.Rect(label_rect.left - 8, label_rect.top - 8, label_rect.width + 16, label_rect.height + 16)
            bg_surface = pygame.Surface((bg_rect.width, bg_rect.height), pygame.SRCALPHA)
            bg_surface.fill(self.label_color)
            screen.blit(bg_surface, (bg_rect.left, bg_rect.top))
            screen.blit(label, label_rect)
```

```python
# Inventory Class
class Inventory:
    # Initialise Inventory with empty item list
    def __init__(self):
        self.items = []

    # Add item to inventory if not already present
    def add(self, item_name):
        if item_name not in self.items: # Check if item already exists
            self.items.append(item_name)

    # Remove item from inventory if it exists
    def remove(self, item_name):
        if item_name in self.items: # Check if item exists before removing
            self.items.remove(item_name)

    # Check if item is in inventory
    def contains(self, item_name):
        return item_name in self.items

    # Clear all items from inventory
    def get_items(self):
        return list(self.items)

    # Draw inventory items on screen
    def draw(self, screen, font, rect, item_size=100, columns=6, padding=20):
        pygame.draw.rect(screen, (30, 30, 30), rect)
        pygame.draw.rect(screen, (255, 255, 255), rect, 2)

        title_text = font.render("Inventory", True, (255, 255, 255))
        title_rect = title_text.get_rect(center=(rect.centerx, rect.top + 30))
        screen.blit(title_text, title_rect)

        start_x = rect.left + padding
        start_y = rect.top + 70
        for index, item in enumerate(self.items): # Iterate through each item in inventory
            row = index // columns
```

```python
class Inventory:
    def draw(self, screen, font, rect, item_size=100, columns=6, padding=20):
            row = index // columns
            col = index % columns
            x = start_x + col * (item_size + padding)
            y = start_y + row * (item_size + padding)

            try: # Load item image and draw on screen
                img = pygame.image.load(f"inventory_items/{item}.png").convert_alpha()
                img = pygame.transform.scale(img, (item_size, item_size))
                screen.blit(img, (x, y))
            except:
                pass
```