
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										

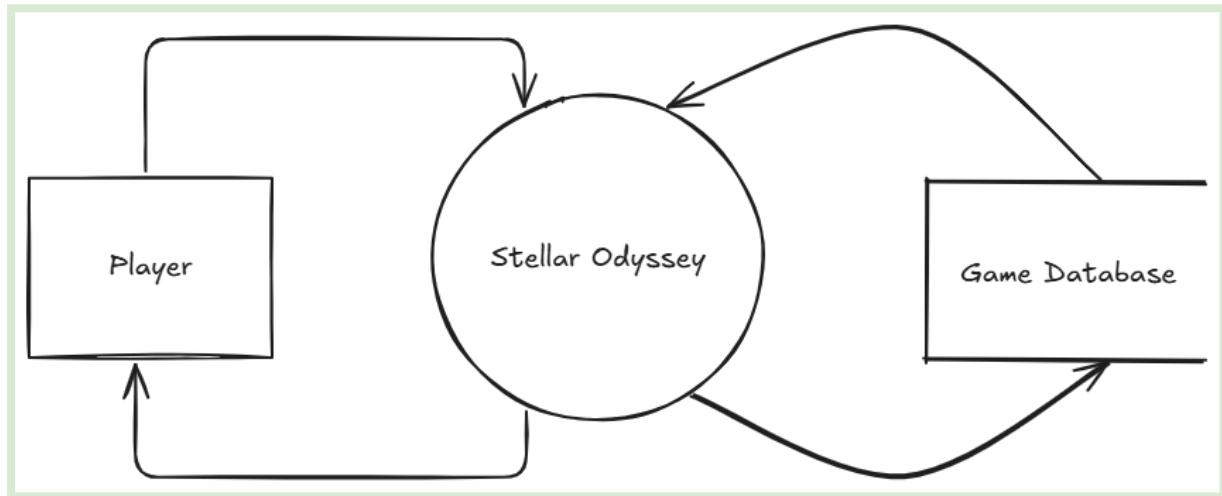
Saving & Security										
Test + Refine Code Data Saving & Security										
Clean Up Code + Small Additions										
Evaluate Code										

→ 2.4 Communication Plan

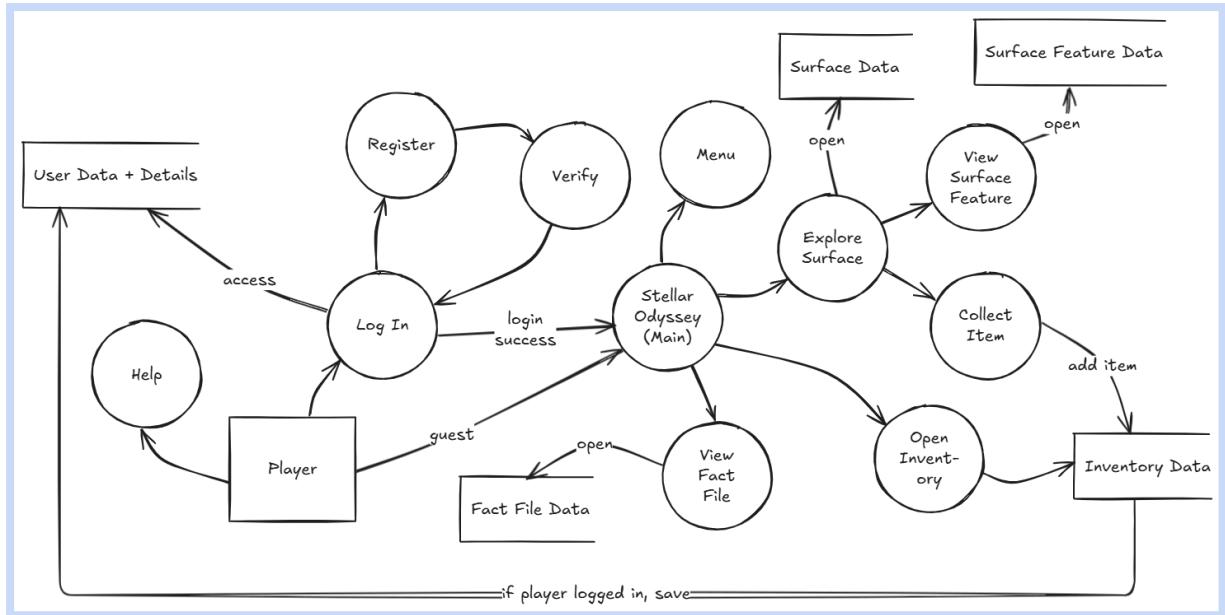
Early development proposals and checkpoints will be submitted for review and guidance. Additionally, informal testing and feedback sessions through my peers will assist in refining gameplay, usability and educational clarity.

3.0 System Design

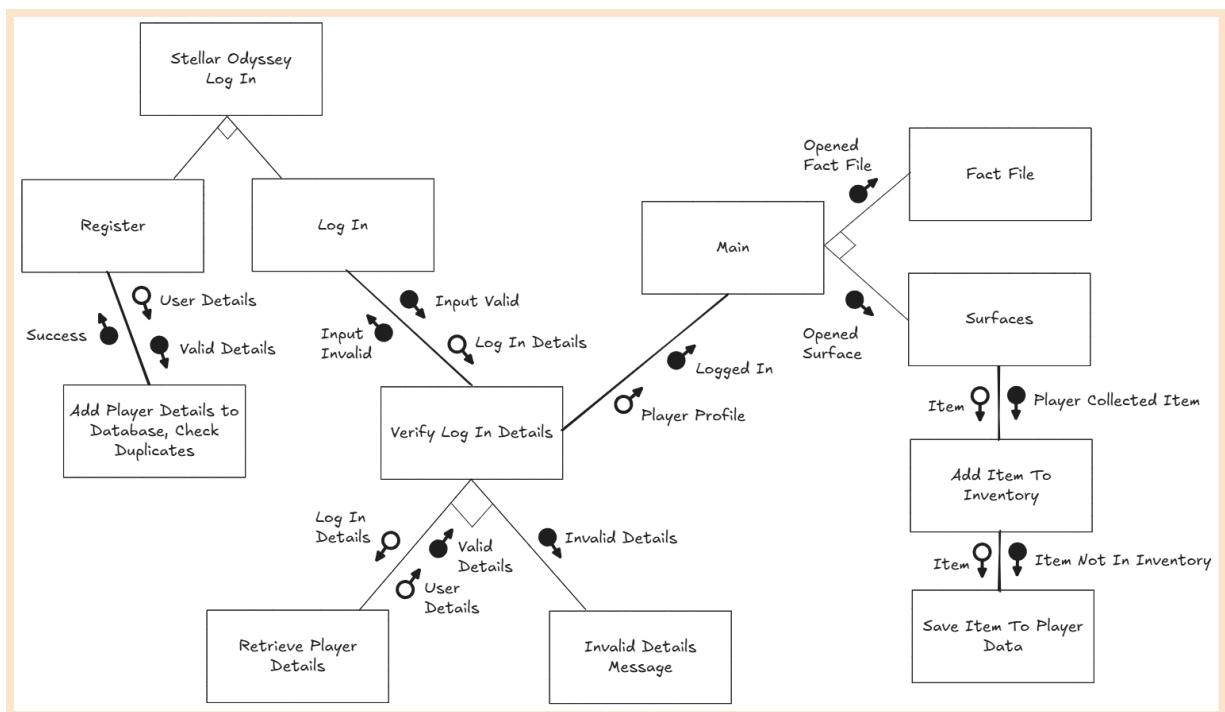
→ 3.1 Context Diagram (Level 0 Data Flow)



→ 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

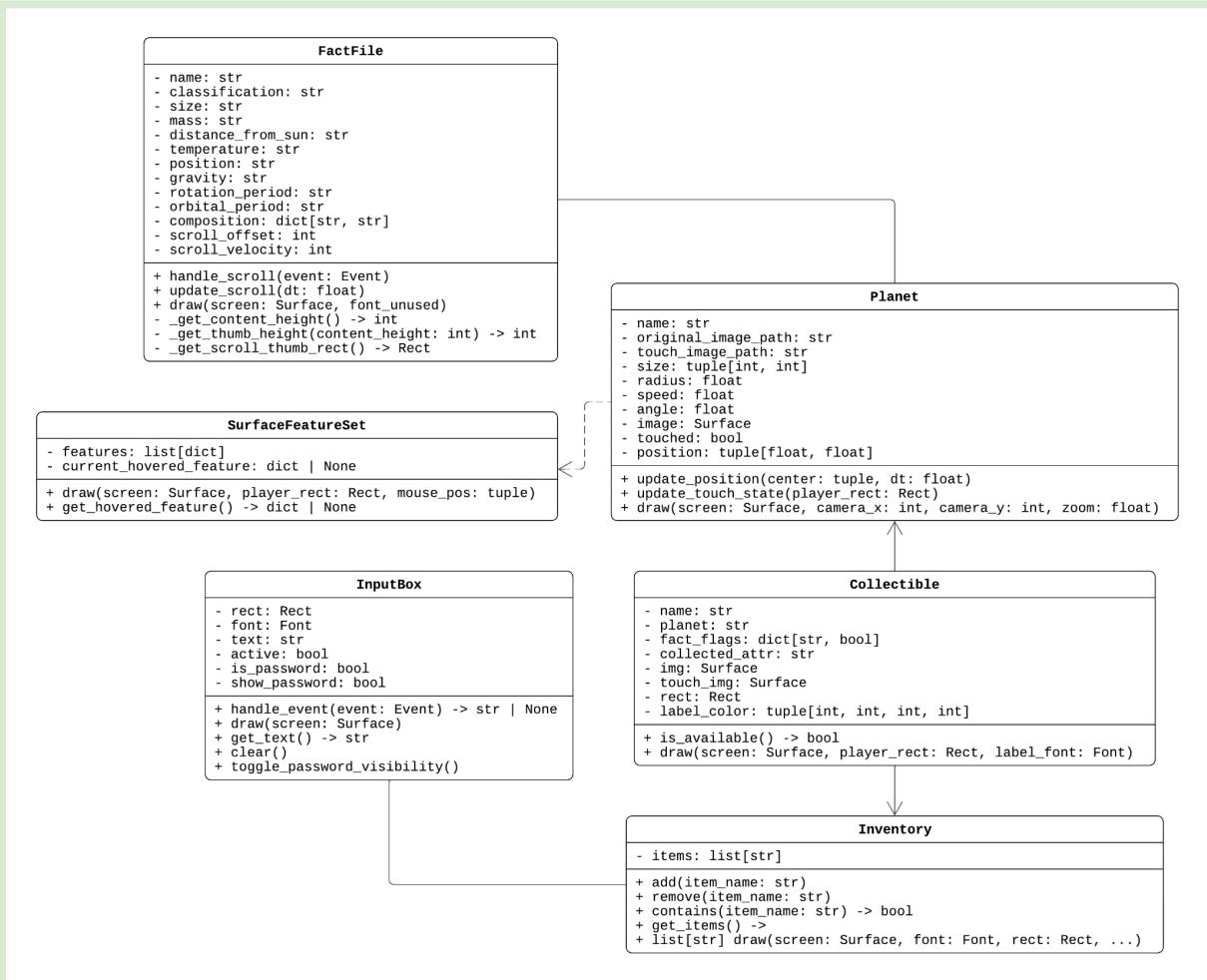
Name	Data Type	Size/Format	Description	Example Value	Constraints/Validation Rules
current_user	str / None	≤ 20 chars	Logged-in user's username	"vin" / None	Alphanumeric only, required to access save/load
player_pos	list[int]	[x: 0–7000, y: 0–7000]	Player's world coordinates	[3500, 3600]	Must remain within 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	"superno va123"	≤ 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.

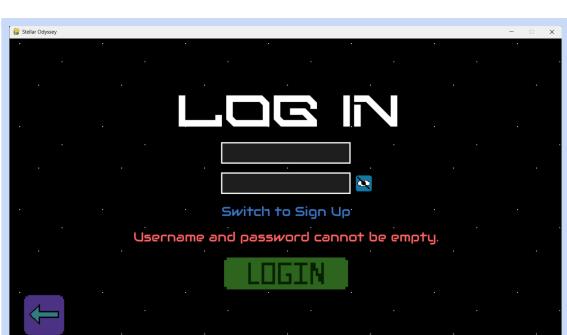
User Interface

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



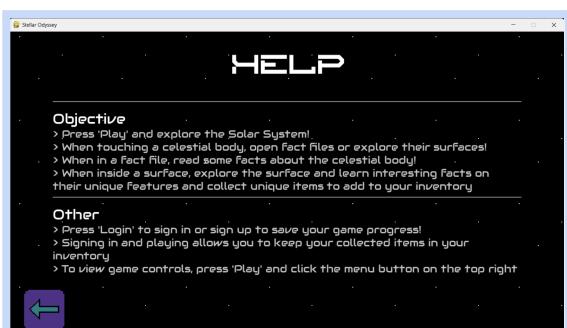
Home Menu: Allows access to play, login and help



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



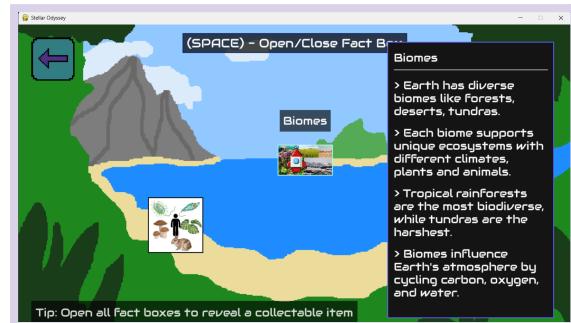
Logged-In Main Game: Main game interface with player profile



Zoom Out: Zoomed out UI



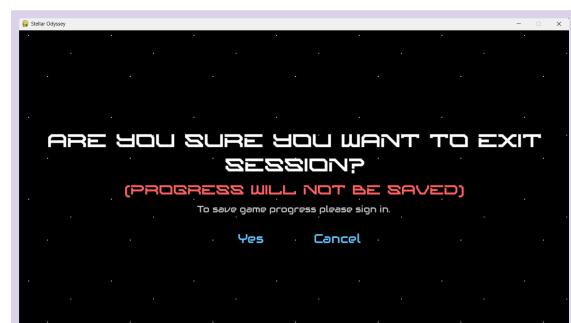
Zoom In: Zoomed in UI



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

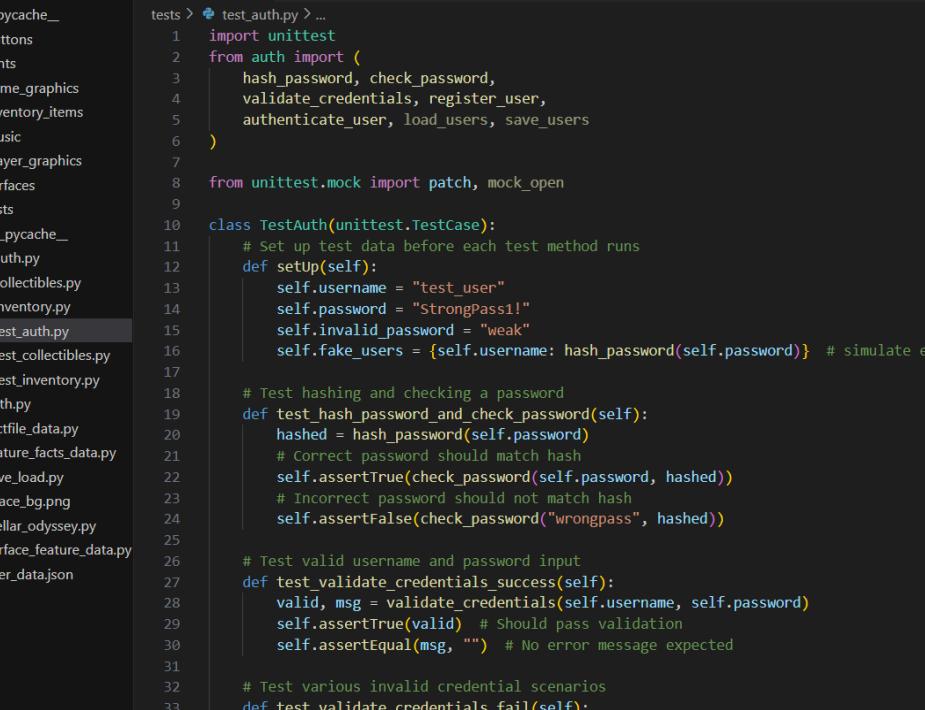
(1) Unit Test (Authentication, Collectible, Inventory)

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 (Authentication Success)	Login succeeds (True)	Authentication works and hash is validated correctly	Pass
UT06	Unit Test (Authentication 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 Item)	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)



The screenshot shows a code editor interface with a dark theme. The left sidebar contains icons for file operations like Open, Save, and Find, along with a tree view of project files. The main area displays a Python test script named `test_auth.py`. The code uses the `unittest` framework to test various authentication functions from the `auth` module.

```
tests > test_auth.py > ...
1 import unittest
2 from auth import (
3     hash_password, check_password,
4     validate_credentials, register_user,
5     authenticate_user, load_users, save_users
6 )
7
8 from unittest.mock import patch, mock_open
9
10 class TestAuth(unittest.TestCase):
11     # Set up test data before each test method runs
12     def setup(self):
13         self.username = "test_user"
14         self.password = "StrongPass1!"
15         self.invalid_password = "weak"
16         self.fake_users = {self.username: hash_password(self.password)} # simulate existing user
17
18     # Test hashing and checking a password
19     def test_hash_password_and_check_password(self):
20         hashed = hash_password(self.password)
21         # Correct password should match hash
22         self.assertTrue(check_password(self.password, hashed))
23         # Incorrect password should not match hash
24         self.assertFalse(check_password("wrongpass", hashed))
25
26     # Test valid username and password input
27     def test_validate_credentials_success(self):
28         valid, msg = validate_credentials(self.username, self.password)
29         self.assertTrue(valid) # Should pass validation
30         self.assertEqual(msg, "") # No error message expected
31
32     # Test various invalid credential scenarios
33     def test_validate_credentials_fail(self):
34         tests = [
35             # Empty fields
36             ("", "", "Username and password cannot be empty."),
37             # Invalid username (too short and invalid characters)
38             ("a", self.password, "Username must be 3-20 characters, and use only letters, numbers, and underscores"),
39             # Invalid password (too short)
40             ("username", "1", "Password must be at least 8 characters long"),
41             # Invalid password (contains invalid characters)
42             ("username", "password!", "Password must contain only letters, numbers, and underscores")
43         ]
44
45         for username, password, error in tests:
46             valid, msg = validate_credentials(username, password)
47             self.assertFalse(valid)
48             self.assertEqual(msg, error)
```

Unit Test Code (Authentication Script)

The screenshot shows a code editor interface with a dark theme. On the left is a sidebar containing project files: __pycache__, buttons, fonts, game_graphics, inventory_items, music, player_graphics, surfaces, tests, __pycache__, auth.py, collectibles.py, inventory.py, test_auth.py (which is selected), test_collectibles.py, test_inventory.py, auth.py, factfile_data.py, feature_facts_data.py, save_load.py, space_bg.png, stellar_odyssey.py, surface_feature_data.py, and user_data.json. The main pane displays the content of test_auth.py:

```
tests > test_auth.py > ...
10 class TestAuth(unittest.TestCase):
11     def test_register_user_success(self, mock_load, mock_save):
12         # Test successful user registration with mocked load/save functions
13         @patch("auth.load_users", return_value={})
14         @patch("auth.save_users")
15         def test_register_user_success(self, mock_load, mock_save):
16             success, msg = register_user("new_user", self.password)
17             self.assertTrue(success)
18             self.assertEqual(msg, "User registered successfully.")
19             # Check that new user was passed to save_users
20             args = mock_save.call_args[0][0]
21             self.assertIn("new_user", args)
22
23         # Test registration fails if username already exists
24         @patch("auth.load_users", return_value={"test_user": "existing_hash"})
25         def test_register_user_already_exists(self, mock_load):
26             success, msg = register_user("test_user", self.password)
27             self.assertFalse(success)
28             self.assertEqual(msg, "Username already exists.")
29
30         # Test successful authentication of a user
31         @patch("auth.load_users")
32         def test_authenticate_user_success(self, mock_load):
33             hashed = hash_password(self.password)
34             mock_load.return_value = {self.username: hashed}
35             self.assertTrue(authenticate_user(self.username, self.password)) # Should pass
36
37         # Test failed authentication due to user not existing
38         @patch("auth.load_users", return_value={})
39         def test_authenticate_user_fail(self, mock_load):
40             self.assertFalse(authenticate_user(self.username, self.password)) # Should fail
41
42     if __name__ == '__main__':
43         unittest.main()
```

Unit Test Code (Authentication Script) #2

The screenshot shows a code editor interface with a dark theme. The left sidebar displays a file tree with several Python files and folders, including `test_auth.py`, `auth.py`, `collectibles.py`, `inventory.py`, and others. The main editor area shows the content of `test_auth.py`:

```
tests > test_auth.py > ...
1 import unittest
2 from auth import (
3     hash_password, check_password,
4     validate_credentials, register_user,
5     authenticate_user, load_users, save_users
6 )
7
8 from unittest.mock import patch, mock_open
9
10 class TestAuth(unittest.TestCase):
11     # Set up test data before each test method runs
12     def setUp(self):
13         self.username = "test_user"
14         self.password = "StrongPass1!"
15         self.invalid_password = "weak"
16         self.fake_users = {self.username: hash_password(self.password)} # simulate existing user
17
18     # Test hashing and checking a password
19     def test_hash_password_and_check_password(self):
20         hashed = hash_password(self.password)
21         # Correct password should match hash
22         self.assertTrue(check_password(self.password, hashed))
23         # Incorrect password should not match hash
24         self.assertFalse(check_password("wrongpass", hashed))
25
26     # Test valid username and password input
27     def test_validate_credentials_success(self):
28         valid, msg = validate_credentials(self.username, self.password)
29         self.assertEqual(valid) # Should pass validation
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
218
219
219
220
221
222
223
224
225
226
227
227
228
229
229
230
231
232
233
234
235
236
237
237
238
239
239
240
241
242
243
244
245
245
246
247
247
248
248
249
249
250
250
251
251
252
252
253
253
254
254
255
255
256
256
257
257
258
258
259
259
260
260
261
261
262
262
263
263
264
264
265
265
266
266
267
267
268
268
269
269
270
270
271
271
272
272
273
273
274
274
275
275
276
276
277
277
278
278
279
279
280
280
281
281
282
282
283
283
284
284
285
285
286
286
287
287
288
288
289
289
290
290
291
291
292
292
293
293
294
294
295
295
296
296
297
297
298
298
299
299
300
300
301
301
302
302
303
303
304
304
305
305
306
306
307
307
308
308
309
309
310
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
```

The terminal below shows the command-line output of the test run:

```
PS C:\Users\dream\Downloads\stellar_odyssey_software_task3 - testing> & C:/Users/dream/AppData/Local/Microsoft/Windows/stellar_odyssey_software_task3 - testing/tests/test_auth.py"
.....
-----
Ran 7 tests in 3.654s
OK
PS C:\Users\dream\Downloads\stellar_odyssey_software_task3 - testing>
```

Unit Test Code (Authentication Script) - Result

The screenshot shows a code editor interface with a dark theme. On the left is a sidebar containing project files and folders. The main area displays a Python script named `test_collectibles.py`. The code is a unit test for a function `is_collectible_available`, which checks if a collectible item is available based on collected facts and its presence in the inventory.

```
tests > test_collectibles.py > ...
1 import unittest
2 from collectibles import is_collectible_available
3
4 class TestCollectibleLogic(unittest.TestCase):
5
6     # Test that the collectible is available when:
7     # - all facts are open (True),
8     # - the item hasn't already been collected (collected = False),
9     # - the item isn't in the inventory yet.
10    def test_item_available_when_all_facts_open_and_not_collected(self):
11        facts = {"a": True, "b": True}
12        collected = False
13        inventory = []
14        result = is_collectible_available(facts, collected, inventory, "Leaf")
15        self.assertTrue(result)
16
17    # Test that the collectible is not available if it has already been marked as collected
18    def test_item_not_available_when_collected_flag_true(self):
19        facts = {"a": True, "b": True}
20        collected = True # Already collected
21        inventory = []
22        result = is_collectible_available(facts, collected, inventory, "Leaf")
23        self.assertFalse(result)
24
25    # Test that the collectible is not available if not all facts are open
26    def test_item_not_available_when_facts_incomplete(self):
27        facts = {"a": True, "b": False} # One fact is not open
28        collected = False
29        inventory = []
30        result = is_collectible_available(facts, collected, inventory, "Leaf")
31        self.assertFalse(result)
32
33    # Test that the collectible is not available if it already exists in the inventory,
34    # regardless of the collected flag
35    def test_item_not_available_if_already_in_inventory(self):
36        facts = {"a": True, "b": True}
37        collected = False
38        inventory = ["leaf"] # Already in inventory, case-insensitive check assumed
```

Unit Test Code (Collectibles Script)

Unit Test Code (Collectibles Script) - Result

The screenshot shows a code editor interface with a dark theme. On the left is a sidebar containing project files and folders: __pycache_, buttons, fonts, game_graphics, inventory_items, music, player_graphics, surfaces, tests, __pycache_, auth.py, collectibles.py, inventory.py, test_auth.py, test_collectibles.py, test_inventory.py (which is the active file), auth.py, factfile_data.py, feature_facts_data.py, save_load.py, space_bg.png, stellar_odessey.py, surface_feature_data.py, and user_data.json. The main pane displays the Python script test_inventory.py:

```
test_inventory.py
tests > test_inventory.py > ...
1 import unittest
2 from inventory import Inventory
3
4 class TestInventory(unittest.TestCase):
5
6     def setUp(self):
7         # Create a new Inventory instance before each test
8         self.inv = Inventory()
9
10    # Test that a new inventory starts empty
11    def test_initial_inventory_is_empty(self):
12        self.assertEqual(self.inv.get_items(), [])
13
14    # Test that adding an item puts it in the inventory
15    def test_add_item(self):
16        self.inv.add("Plasma")
17        self.assertIn("Plasma", self.inv.get_items())
18
19    # Test that adding a duplicate item doesn't add it twice
20    def test_add_duplicate_item(self):
21        self.inv.add("Plasma")
22        self.inv.add("Plasma") # Try to add again
23        self.assertEqual(len(self.inv.get_items()), 1)
24
25    # Test that removing an item deletes it from the inventory
26    def test_remove_item(self):
27        self.inv.add("Leaf")
28        self.inv.remove("Leaf")
29        self.assertNotIn("Leaf", self.inv.get_items())
30
31    # Test that removing a nonexistent item doesn't crash and has no effect
32    def test_remove_nonexistent_item(self):
33        self.inv.remove("Basalt") # Item was never added
34        self.assertEqual(self.inv.get_items(), [])
35
36    # Test the contains() method correctly identifies present and absent items
37    def test_contains_item(self):
38        self.inv.add("Blueberries")
```

Unit Test Code (Inventory Script)

The screenshot shows a code editor interface with a dark theme. On the left is a sidebar with icons for file operations like Open, Save, and Find, and a tree view of project files. The main area is a code editor with a search bar at the top right. The search bar contains the text "stellar_odyssey_software_task3 - testing". The code editor displays a Python file named "test_inventory.py". The code is a unit test for an "Inventory" class, using the "unittest" module. It includes tests for the "contains" method, the return type of "get_items()", and the main block. The code is color-coded for readability.

```
tests > test_inventory.py > TestInventory
4  class TestInventory(unittest.TestCase):
5      def test_contains_item(self):
6          self.inv.add("Blueberries")
7          self.assertTrue(self.inv.contains("Blueberries")) # Should be found
8          self.assertFalse(self.inv.contains("Rock")) # Should not be found
9
10         # Test that get_items() returns a copy, not a reference
11         def test_get_items_returns_copy(self):
12             self.inv.add("Water Ice")
13             copy = self.inv.get_items()
14             copy.append("Fake") # Modify the copy
15             # The original inventory should not be affected
16             self.assertNotIn("Fake", self.inv.get_items())
17
18         if __name__ == '__main__':
19             unittest.main()
```

Unit Test Code (Inventory Script) #2

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows a project structure with files like `__pycache__`, `buttons`, `fonts`, `game_graphics`, `inventory_items`, `music`, `player_graphics`, `surfaces`, and several test files (`tests` folder containing `auth.py`, `collectibles.py`, `inventory.py`, `test_auth.py`, `test_collectibles.py`, `test_inventory.py` which is selected, `factfile_data.py`, `feature_facts_data.py`, `save_load.py`, `space_bg.png`, `stellar_odyssey.py`, `surface_feature_data.py`, and `user_data.json`).
- Code Editor (Center):** Displays the content of `test_inventory.py`. The code uses `unittest` to test an `Inventory` class. It includes tests for initial state, adding items, duplicates, and removal.
- Terminal (Bottom):** Shows the output of a `pygame` run, indicating it's from the community. It also shows the results of a test run: "Ran 7 tests in 0.000s" and "OK".

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	Revisit Surface	Collectible does not reappear	No Blueberries on surface	Pass

Integration Test (In-Game Examples)

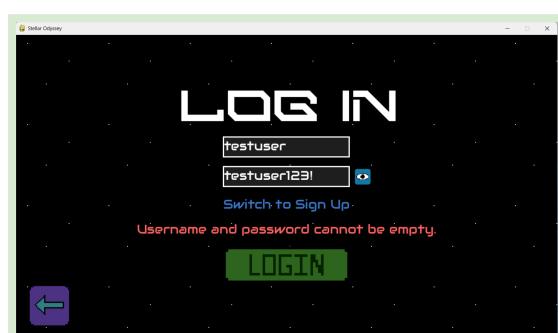
<p>'testuser' player credentials created in sign up page successfully</p>	<p>'testuser' player profile successfully appears (top left)</p>



'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

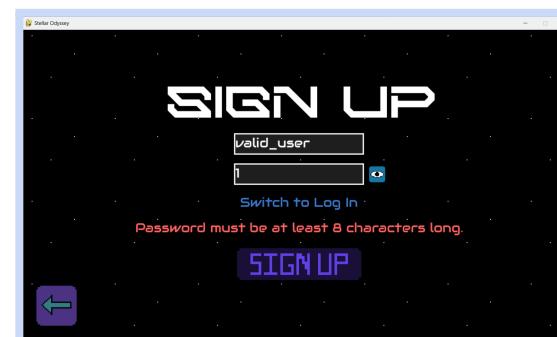
Overall all main components of Login, Inventory Persistence and Collectible Logic all work together properly and successfully.

(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)

 <p>Test ID: TD01</p>	 <p>Test ID: TD02</p>
--	---



→ 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, minigame 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:

- **Programming:** 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)

```

197     """Stellar Odyssey - Classes"""
198
199     # SurfaceFeatureSet Class
200     class SurfaceFeatureSet:
201         # Initialise SurfaceFeatureSet with list of features
202         def __init__(self, features):
203             self.features = []
204             for feature in features: # Load each feature's normal and touch images
205                 normal_img = pygame.image.load(f"game_graphics/surface_graphics/{feature['name']}.png").convert_alpha()
206                 touch_img = pygame.image.load(f"game_graphics/surface_graphics/{feature['name']}_touch.png").convert_alpha()
207                 rect = normal_img.get_rect(topleft=feature["pos"])
208                 self.features.append({
209                     "name": feature["name"],
210                     "image": normal_img,
211                     "touch_image": touch_img,
212                     "rect": rect,
213                     "hovered": False
214                 })
215             self.current_hovered_feature = None
216
217         # Draw all features on screen, highlighting hovered feature
218         def draw(self, screen, player_rect, mouse_pos):
219             self.current_hovered_feature = None
220             for feature in self.features: # Iterate through each feature
221                 is_touching = player_rect.colliderect(feature["rect"])
222                 is_hovering = feature["rect"].collidepoint(mouse_pos)
223                 feature["hovered"] = is_touching or is_hovering
224
225                 current_img = feature["touch_image"] if feature["hovered"] else feature["image"]
226                 screen.blit(current_img, feature["rect"].topleft)
227
228                 if feature["hovered"]: # If feature is hovered or touched, display its label
229                     self.current_hovered_feature = feature
230                     display_name = feature["name"].replace("_", " ").title()
231                     label = surface_label_font.render(display_name, True, (255, 255, 255))
232                     label_rect = label.get_rect(midbottom=(feature["rect"].centerx, feature["rect"].top - 10))

```

```

200     class SurfaceFeatureSet:
201         def draw(self, screen, player_rect, mouse_pos):
202
203             padding = 8
204             bg_rect = pygame.Rect(
205                 label_rect.left - padding,
206                 label_rect.top - padding,
207                 label_rect.width + 2 * padding,
208                 label_rect.height + 2 * padding
209             )
210             bg_surface = pygame.Surface((bg_rect.width, bg_rect.height), pygame.SRCALPHA)
211             bg_surface.fill((0, 0, 0, 160))
212             screen.blit(bg_surface, (bg_rect.left, bg_rect.top))
213
214             screen.blit(label, label_rect)
215
216             # Handle mouse events for features
217             def get_hovered_feature(self):
218                 return self.current_hovered_feature

```

```

251 # InputBox Class
252 class InputBox:
253     # Initialise InputBox with position, size, font and password mode
254     def __init__(self, x, y, w, h, font, is_password=False):
255         self.rect = pygame.Rect(x, y, w, h)
256         self.font = font
257         self.color_inactive = pygame.Color('white')
258         self.color_active = pygame.Color('deepskyblue')
259         self.color_hover = pygame.Color('lightskyblue')
260         self.color = self.color_inactive
261         self.text = ''
262         self.txt_surface = font.render('', True, self.color)
263         self.active = False
264         self.is_password = is_password
265         self.show_password = False
266         self.scroll_offset = 0
267
268     # Handle events for the input box
269     def handle_event(self, event):
270         if event.type == pygame.MOUSEBUTTONDOWN:
271             self.active = self.rect.collidepoint(event.pos) # Check if clicked inside the box
272         if event.type == pygame.KEYDOWN and self.active:
273             if event.key == pygame.K_RETURN: # Continue when 'Enter' key pressed
274                 return 'submit'
275             elif event.key == pygame.K_BACKSPACE: # Remove character when backspace pressed
276                 self.text = self.text[:-1]
277             else: # Handle other keys
278                 if len(self.text) < 20:
279                     self.text += event.unicode
280             if self.is_password: # If password mode, update display text
281                 display_text = self.text if self.show_password else '*' * len(self.text)
282             else:
283                 display_text = self.text
284             self.txt_surface = self.font.render(display_text, True, self.color)
285
286             text_width = self.txt_surface.get_width()

```

```

252 class InputBox:
253     def handle_event(self, event):
254         text_width = self.txt_surface.get_width()
255         if text_width > self.rect.width - 20: # If text exceeds box width, enable scrolling
256             self.scroll_offset = text_width - (self.rect.width - 20)
257         else:
258             self.scroll_offset = 0
259
260         return None
261
262     # Update input box state
263     def draw(self, screen):
264         mouse_pos = pygame.mouse.get_pos()
265         if self.rect.collidepoint(mouse_pos): # Check if mouse is hovering over box
266             if not self.active: # If box not active, change colour to hover colour
267                 border_color = self.color_hover
268             else:
269                 border_color = self.color_active
270         else:
271             border_color = self.color_active if self.active else self.color_inactive
272
273         pygame.draw.rect(screen, (30, 30, 30), self.rect)
274         pygame.draw.rect(screen, border_color, self.rect, 3)
275
276         # Create surface for text and fill with a dark colour
277         text_clip_surface = pygame.Surface((self.rect.width - 6, self.rect.height - 6))
278         text_clip_surface.fill((30, 30, 30))
279
280         text_clip_surface.blit(self.txt_surface, (-self.scroll_offset, 0))
281
282         screen.blit(text_clip_surface, (self.rect.x + 3, self.rect.y + 3))
283
284     # Get current text in input box
285     def get_text(self):
286         return self.text

```

```

252 class InputBox:
253
254     # Clear input box text
255     def clear(self):
256         self.text = ''
257         self.txt_surface = self.font.render('', True, self.color)
258
259     # Toggle password visibility
260     def toggle_password_visibility(self):
261         if self.is_password: # Only toggle if in password mode
262             self.show_password = not self.show_password
263             display_text = self.text if self.show_password else '*' * len(self.text)
264             self.txt_surface = self.font.render(display_text, True, self.color)

```

```

332 # FactFile Class
333 class FactFile:
334     # Initialise FactFile with planet data
335     def __init__(self, name, classification, size, mass, distance_from_sun, temperature,
336                  position, gravity, rotation_period, orbital_period, composition=None):
337         self.name = name
338         self.classification = classification
339         self.size = size
340         self.mass = mass
341         self.distance_from_sun = distance_from_sun
342         self.temperature = temperature
343         self.position = position
344         self.gravity = gravity
345         self.rotation_period = rotation_period
346         self.orbital_period = orbital_period
347         self.scroll_offset = 0
348         self.scroll_velocity = 0
349         self.scroll_speed = 600
350         self.scroll_direction = 0
351         self.dragging_scrollbar = False
352         self.drag_start_y = 0
353         self.initial_scroll_offset = 0
354         self.composition = composition or {}
355
356     # Handle scroll events for fact file
357     def handle_scroll(self, event):
358         if event.type == pygame.KEYDOWN:
359             if event.key == pygame.K_UP: # Scroll up when 'Up' key pressed
360                 self.scroll_direction = 1
361             elif event.key == pygame.K_DOWN: # Scroll down when 'Down' key pressed
362                 self.scroll_direction = -1
363             elif event.type == pygame.KEYUP: # Stop scrolling when key released
364                 if event.key in (pygame.K_UP, pygame.K_DOWN):
365                     self.scroll_direction = 0

```

```

333     class FactFile:
334
335         def handle_scroll(self, event):
336             self.scroll_direction = 0
337
338             elif event.type == pygame.MOUSEBUTTONDOWN:
339                 if event.button == 4: # Scroll up with mouse wheel
340                     self.scroll_offset += 60
341                 elif event.button == 5: # Scroll down with mouse wheel
342                     self.scroll_offset -= 60
343                 elif event.button == 1: # Left mouse button pressed
344                     mouse_x, mouse_y = event.pos
345                     thumb_rect = self._get_scroll_thumb_rect()
346                     if thumb_rect.collidepoint(mouse_x, mouse_y): # Check if clicked on scrollbar thumb
347                         self.dragging_scrollbar = True
348                         self.drag_start_y = mouse_y
349                         self.initial_scroll_offset = self.scroll_offset
350
351             elif event.type == pygame.MOUSEBUTTONUP: # Release mouse button
352                 if event.button == 1:
353                     self.dragging_scrollbar = False
354
355             elif event.type == pygame.MOUSEMOTION: # Mouse moved
356                 if self.dragging_scrollbar: # If dragging scrollbar
357                     dy = event.pos[1] - self.drag_start_y
358                     content_height = self._get_content_height()
359                     visible_height = HEIGHT // 2
360                     max_offset = max(1, content_height - visible_height)
361                     scrollable_area = visible_height - self._get_thumb_height(content_height)
362                     if scrollable_area > 0: # Prevent division by zero
363                         scroll_ratio = dy / scrollable_area
364                         self.scroll_offset = self.initial_scroll_offset - scroll_ratio * max_offset
365
366             # Update scroll offset based on scroll speed and direction
367             def update_scroll(self, dt):
368                 self.scroll_velocity = self.scroll_speed * self.scroll_direction
369                 self.scroll_offset += self.scroll_velocity * dt
370                 self.scroll_offset = min(0, self.scroll_offset)

```

```

333 class FactFile:
334
335     # Draw fact file on screen
336     def draw(self, screen):
337         draw_parallax_bg()
338
339         # Set up
340         name_font = pygame.font.Font("fonts/stellar.otf", 80)
341         content_font = default_font
342         composition_font = pygame.font.Font("fonts/audiowide.ttf", 36)
343
344         # Render Fact File Content
345         lines = [
346             f"{self.name}",
347             f"Classification: {self.classification}",
348             f"Position in Solar System: {self.position}",
349             f"Distance from Sun: {self.distance_from_sun}",
350             f"Diameter: {self.size}",
351             f"Mass: {self.mass}",
352             f"Surface Gravity: {self.gravity}",
353             f"Rotation Period: {self.rotation_period}",
354             f"Orbital Period: {self.orbital_period}",
355             f"Average Surface Temperature: {self.temperature}"
356         ]
357
358         title_text = factfile_name_font.render(lines[0], True, (255, 255, 255))
359
360         line_spacing = 20
361         start_y = HEIGHT // 4 + self.scroll_offset
362
363         for line in lines[1:]: # Render each line of fact file
364             text_surface = factfile_content_font.render(line, True, (255, 255, 255))
365             faded_surface = factfile_content_font.render(line, True, (180, 180, 180))
366
367             content_height = self._get_content_height()
368             visible_height = HEIGHT // 2
369             min_scroll = min(0, visible_height - content_height)

```

```

333 class FactFile:
402     def draw(self, screen):
436         if self.scroll_offset < min_scroll: # Clamp scroll offset to minimum
437             self.scroll_offset = min_scroll
438         elif self.scroll_offset > 0: # Clamp scroll offset to maximum
439             self.scroll_offset = 0
440
441         # 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()

```

```

133 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()

```

```

550 # Planet Class
551 class Planet:
552     # Initialise Planet with name, image path, size, radius, speed and angle
553     def __init__(self, name, image_path, size, radius, speed, angle=None):
554         self.name = name
555         self.original_image_path = image_path
556         self.touch_image_path = image_path.replace(".png", "_touch.png")
557         self.size = size
558         self.radius = radius
559         self.speed = speed
560         self.angle = angle if angle is not None else random.uniform(0, 2 * math.pi)
561         self.image = pygame.transform.scale(pygame.image.load(image_path).convert_alpha(), size)
562         self.original_image = self.image.copy()
563         self.touched = False
564         self.position = (0, 0)
565
566     # Update planet position based on angle and speed
567     def update_position(self, center, dt):
568         self.angle += self.speed * dt
569         cx, cy = center
570         self.position = (
571             cx + math.cos(self.angle) * self.radius - self.size[0] // 2,
572             cy + math.sin(self.angle) * self.radius - self.size[1] // 2
573         )
574
575     # Update touch state based on player rectangle
576     def update_touch_state(self, player_rect):
577         planet_rect = pygame.Rect(self.position[0], self.position[1], self.size[0], self.size[1])
578         if planet_rect.colliderect(player_rect): # Check if player touching planet
579             if not self.touched: # If not already touched, change image to touch image
580                 try:
581                     self.image = pygame.transform.scale(pygame.image.load(self.touch_image_path).convert_alpha(), self.size)
582                 except:
583                     pass
584             self.touched = True
585         else: # If player not touching planet, reset image to original
586             if self.touched:

```

```

551     class Planet:
552         def update_touch_state(self, player_rect):
553             if self.touched:
554                 self.image = self.original_image.copy()
555                 self.touched = False
556
557             # Draw planet on screen at its current position
558             def draw(self, screen, camera_x, camera_y, zoom):
559                 scaled = pygame.transform.smoothscale(self.image, (int(self.size[0] * zoom), int(self.size[1] * zoom)))
560                 screen.blit(scaled, ((self.position[0] - camera_x) * zoom, (self.position[1] - camera_y) * zoom))
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593

```

```

596 class Collectible:
598     def __init__(self, name, planet, fact_flags, collected_attr, img, touch_img, rect, label_color):
599         self.name = name
600         self.planet = planet
601         self.fact_flags = fact_flags
602         self.collected_attr = collected_attr
603         self.img = img
604         self.touch_img = touch_img
605         self.rect = rect
606         self.label_color = label_color
607
608     # Check if collectible is available to collect
609     def is_available(self):
610         if all(self.fact_flags.values()) and not getattr(sys.modules[__name__], self.collected_attr): # Check if all fact flags are true and col-
611             normalized_inventory = [item.lower().replace(" ", "_") for item in player_inventory.get_items()]
612             normalized_name = self.name.lower().replace(" ", "_")
613             return normalized_name not in normalized_inventory
614
615     return False
616
617     # Draw collectible on screen
618     def draw(self, screen, player_rect, label_font):
619         if not self.is_available(): # If collectible not available, do not draw
620             return
621
622         hovered = player_rect.colliderect(self.rect)
623         screen.blit(self.touch_img if hovered else self.img, self.rect.topleft)
624
625         if hovered: # If collectible is hovered, display its label
626             label = label_font.render(self.name, True, (255, 255, 255))
627             label_rect = label.get_rect(midbottom=(self.rect.centerx, self.rect.top - 10))
628             bg_rect = pygame.Rect(label_rect.left - 8, label_rect.top - 8, label_rect.width + 16, label_rect.height + 16)
629             bg_surface = pygame.Surface((bg_rect.width, bg_rect.height), pygame.SRCALPHA)
630             bg_surface.fill(self.label_color)
631             screen.blit(bg_surface, (bg_rect.left, bg_rect.top))
632             screen.blit(label, label_rect)

```

```

634 # Inventory Class
635 class Inventory:
636     # Initialise Inventory with empty item list
637     def __init__(self):
638         self.items = []
639
640     # Add item to inventory if not already present
641     def add(self, item_name):
642         if item_name not in self.items: # Check if item already exists
643             self.items.append(item_name)
644
645     # Remove item from inventory if it exists
646     def remove(self, item_name):
647         if item_name in self.items: # Check if item exists before removing
648             self.items.remove(item_name)
649
650     # Check if item is in inventory
651     def contains(self, item_name):
652         return item_name in self.items
653
654     # Clear all items from inventory
655     def get_items(self):
656         return list(self.items)
657
658     # Draw inventory items on screen
659     def draw(self, screen, font, rect, item_size=100, columns=6, padding=20):
660         pygame.draw.rect(screen, (30, 30, 30), rect)
661         pygame.draw.rect(screen, (255, 255, 255), rect, 2)
662
663         title_text = font.render("Inventory", True, (255, 255, 255))
664         title_rect = title_text.get_rect(center=(rect.centerx, rect.top + 30))
665         screen.blit(title_text, title_rect)
666
667         start_x = rect.left + padding
668         start_y = rect.top + 70
669         for index, item in enumerate(self.items): # Iterate through each item in inventory
670             row = index // columns

```

```

635 class Inventory:
636     def draw(self, screen, font, rect, item_size=100, columns=6, padding=20):
637         row = index // columns
638         col = index % columns
639         x = start_x + col * (item_size + padding)
640         y = start_y + row * (item_size + padding)
641
642         try: # Load item image and draw on screen
643             img = pygame.image.load(f"inventory_items/{item}.png").convert_alpha()
644             img = pygame.transform.scale(img, (item_size, item_size))
645             screen.blit(img, (x, y))
646         except:
647             pass

```