The Stratobus Mission Game Report

Group Members:

- Kirsty McVean
- Natalie Martin
- Perrine Hemmings
- Louise Berridge
- Rea Bartlett Tandon
- Alice Morgan

Introduction

In collaboration, Group 6 developed the *Stratobus Mission,* an educational and interactive space-themed game using Python and the PyGame library. The game is designed to engage learners aged 7-12 years, enabling them to operate Thales' Stratobus - a high-altitude, long-endurance stratospheric airship - through virtual cosmos while learning about and completing educational tasks linked to space, weather and near-Earth objects. The game incorporates real-time API data sourced from NASA's Open API Portal, providing players with realistic and engaging space-related challenges. Additionally, a SQL database stores trivia questions and player scores while the PyGame library is employed for rendering diverse environments, interfaces, and challenges, creating an engaging and immersive user experience.

Our team is built on foundations of collaboration, mutual support and a shared determination to showcase our capabilities to our sponsor company, Thales. By understanding our individual strengths and areas for growth, we strategically paired team members to foster learning and expertise exchange, and used constructive peer review to further enhance our work quality and cohesion.

Background

Problem Addressed

The *Stratobus Mission* aims to educate players aged 7-12 years about space concepts and the aerospace industry in an engaging and fun manner. As all group members are sponsored by Thales, we aimed to align with Thales' operations while integrating education and community outreach - appealing attributes that drew us all towards Thales. We imagined ourselves in a real-life scenario where we had been tasked with building an application that not only imparts knowledge about Thales' work but also encourages young learners through active engagement. Our project addresses the challenge of making education enjoyable and immersive in complex areas such as aerospace while exploring a visually stimulating virtual space environment with the sense of achievement that comes from completing tasks. We admire Thales' ethos of fostering volunteering opportunities and giving back to the community, and wanted to build something that would contribute to their corporate social responsibility initiatives.

The time constraint for project planning, design, execution and testing was our major challenge, and for this reason we decided on a modular project to which we could add as many missions as time permitted. We strategically set a baseline target of four challenges to implement to a high standard and work on as priority, then planned to use any remaining time to add further challenges. We successfully met this goal and even incorporated an additional asteroid-blasting game that a team member developed while learning to use the PyGame library.

The *Stratobus Mission* comprises seven distinct missions, each designed to engage, educate, and entertain players while aligning with the Key Stage 2 curriculum:

1. **Asteroid Challenge:** Players use real-time NASA API data to track asteroids' proximity to Earth, while honing their maths skills by rounding distances to the nearest whole number. The player must type in the answer to the question and then click submit.
2. **Satellite Images:** Players harness the Stratobus' satellite links to capture images of cities, enhancing geographical and weather-related knowledge. The player has to answer the question presented by clicking on the correct image.
3. **Capture Mars:** Players control the Mars Rover, switching between the front hazard camera, the rear hazard camera and the navigation camera to view real images of Mars, fostering a connection to interplanetary exploration.
4. **Payload Mission:** Players must optimise data storage by packing data packets with tetrominoes. As the tetrominoes fall, the player must move them using the arrow keys or rotate the tetromino. The aim of the game is to complete as many full rows to clear storage space before the tetrominoes reach the top. This supports KS2 geometry, spatial reasoning and problem solving.
5. **Locate ISS:** Players test their geography skills by tracking the International Space Station's orbit around Earth. Please note that if "Loading" is displayed, the ISS is currently somewhere over water.
6. **Asteroid Blast Mission:** Players navigate their rocket through space, skillfully dodging asteroids using the arrow keys or the spacebar to fire at the asteroids to reach new heights in the game.
7. **Quiz:** A knowledge enhancing challenge, where players answer ten randomised multichoice questions linked to space, maths, and science, followed by the chance to enter their name and see if they have made it onto the leaderboard.

Specifications & Design

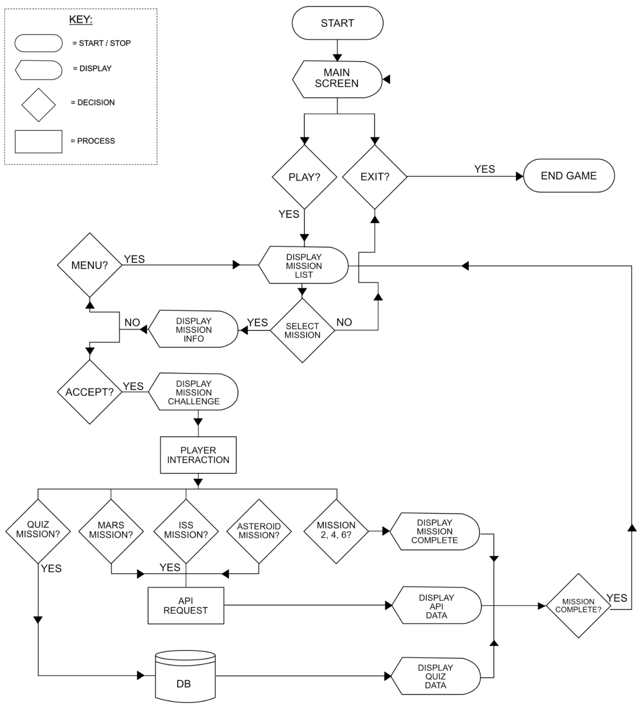
Functional requirements:

1. **User Interaction and Navigation:**
   - All buttons must change colour when the mouse hovers over it and play a clicking sound when the button is clicked to acknowledge the user's response.
   - A mute button in the top right corner should allow players to control the space-themed background music throughout the program being run.
   - The program must feature a welcome homepage with a button for entering the game and another for quitting the program.
   - Once the user has entered the game, a comprehensive menu should display all missions and provide an exit button.
2. **Mission Instructions and Interface:**
   - Each mission should load an instructions page immediately, featuring a button to accept and proceed to the game as well as a menu button for returning to the main menu.
   - In the Asteroid Challenge, players should be able to type their answers in a user input box and submit them for evaluation - feedback should display once the player has submitted their answer.
   - In the Capture Mars mission, buttons to switch between different camera options with immediate image updates should enhance the immersive experience.
   - The Payload Mission should allow players to manoeuvre tetrominoes using arrow keys.
3. **Mission Variety and User Engagement:**
   - A minimum of four challenges should be implemented to provide users with a diverse range of engaging gameplay.
   - The user interface must be visually stimulating featuring colourful, child-friendly icons.
   - Interactive buttons throughout the game should provide engagement and participation.
   - An engaging and legible typewriter font that simulates letter-by-letter typing enhancing the space theme, encouraging sustained engagement as well as promoting readability for young players.
   - API data integration should ensure real-life content to provide authentic content.
   - The SQL database should store trivia questions as well as data for the leaderboard.

Non-functional requirements:

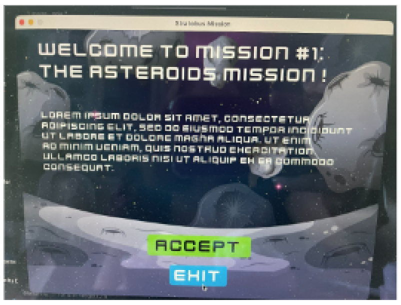
1. **Performance and User Experience:**
   - Loading times for missions should be optimised to maintain a smooth user experience, with a maximum duration of 5 seconds.
   - The user interface should be intuitive and user-friendly.
   - The game should perform on both a Windows and Mac computer.
2. **Reliability and Stability:**
   - The game should run without crashes or glitches, ensuring stability and reliability for users.
3. **Usability and Accessibility:**
   - The game's visual and auditory elements as well as use of language should be adaptable to the ability and sensory needs of children aged 7-12.
4. **Educational Value and Awareness:**
   - The game should align with the Key Stage 2 curriculum, imparting educational content through engaging challenges, as well as serving as a platform to promote the work and values of Thales and their commitment to education, innovation, and community engagement.

Architecture Diagram:

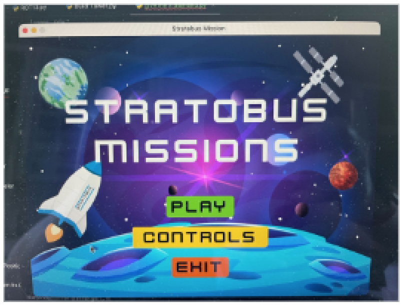


Design:

A captivating visual experience was crucial for engaging our young target audience. Our graphic design expert developed a space-themed series of engaging and vibrant scene backgrounds for PyGame implementation. We decided on a colourful, cartoon-ish style with a futuristic font that reflected the wonder and high-tech element that space travel represented for all of us.


WELCOME TO MISSION #1: THE ASTEROIDS MISSION!

LOREM IPSUM DOLOR SIT AMET, CONSECTETUR ADIPISCING ELIT, SED DO EIUSMOD TEMPOR INCIDIDUNT UT LABORE ET DOLORE MAGNA ALIQUA. UT ENIM AD MINIM VENIAM, QUIS NOSTRUD EXERCITATION ULLAMCO LABORIS NISI UT ALIQUIP EX EA COMMODO CONSEQUAT.



Early-stage design concepts

The program uses a modular structure to make the code reusable, easy to work with, maintainable and scalable. Each mission is placed in its own separate Python file in order to apply efficient unit testing. This enabled team members to simultaneously work on different areas of the program. We followed Object-Oriented Programming principles to efficiently organise our Python code making it readable, consistent and avoid code redundancy. This design keeps the management of the project easy to sustain and supports good teamwork.

Implementation & Execution

Development approach and team member roles:

Team collaboration was integral, reflecting real-world software development practices, and we successfully banded together to support each other and work through challenges effectively. We used pair programming where possible and followed agile methodologies to deliver our final project to a standard we are proud of.

We used Trello for efficient ticket management and held short daily stand-ups to motivate each other and keep ourselves accountable for progress. In addition to this, we met approximately twice per week (ad-hoc) on longer calls to discuss ideas and overcome obstacles together. We were mindful that not all group members would be able to attend all stand-ups, so to mitigate this we recorded minutes, attendees and actions for all our meetings.

All code and supplementary files were managed via Github, with development branches that were peer reviewed before merging. Discord helped us to stay connected and Mural-facilitated collaboration and idea sharing. Our approach fostered an environment of teamwork where we shared knowledge and pushed one another to succeed, resulting in a successful project outcome. Effective communication and skill-building were our guiding principles, driving the success of the *Stratobus Mission*.

The main tasks were divided as follows:

- **All:** PyGame, unit testing, user testing, refactoring, docstring writing, report writing, paired programming, GitHub, ideation, collaborative problem solving
- **Alice:** SQL trivia & scoreboard databases, Python <> DB connection, README doc, project report
- **Kirsty:** Lead PyGame implementation, facilitating integration, sourced music, refactoring, utilities.
- **Louise:** PyGame implementation, Asteroid Blast challenge
- **Natalie:** Payload challenge, requirements/library install
- **Perrine:** Graphic and asset design, Mars Rover challenge, ISS challenge
- **Rea:** Asteroids challenge, Satellite Images challenge + PyGame implementation, project report

Tools and libraries:

PyGame
Central to our project, the PyGame library brings together diverse elements and renders them into a user-friendly interface. Our PyGame implementation brings Python-based challenges to life with an original and striking visual design, incorporating background music for added impact for the end user. The framework includes interactive buttons and text input fields for user interaction. Extensive research of the PyGame library was required to complete this project to a high standard, and the more we learned about PyGame and its requirements and limitations, the more we had to adapt our code-writing processes in an agile way.

Python Code
We followed Object-Oriented Programming principles to efficiently organise our Python code and avoid code redundancy.

Python Libraries
The following Python libraries were used in our project to enhance functionality:

- PyGame
- DateTime
- Requests
- GeoPy
- Random

- Googletrans
- Math
- Aiohttp
- Asyncio

Third-party APIs

The following external APIs were used to pull real-time data to our game interface:

- NeoWs (Near-Earth Objects) - NASA OpenAPI
- Sentinel Hub Process API (Sentinel Hub)
- Mars Rover Photos (NASA OpenAPI)
- International Space Station Location (Open Notify API)

SQL Database

A SQL database stored trivia questions and answers, contributing to efficient data management. Randomised question order added dynamism, while a separate table tracked player scores for the leaderboard.

Implementation process:

We opted to use PyGame early on, leveraging a team member's previous experience with this library. However, a much deeper understanding of PyGame was required to complete this project to a high standard, which entailed extensive research and a steep learning curve for those working on the PyGame implementation. As a group, we learned to adapt our coding style for implementation into the PyGame library (for example different handling of user input, far more modular approach, no use of "print" statements). We had to adapt the architecture and outputs of the code of our individual challenges to effectively render outputs in PyGame and capture user input. This resulted in extensive code refactoring and required clear communication between the team. Despite the challenges we faced, our shared commitment to quickly mastering this skill brought about satisfying results.

A fundamental aspect of our project was incorporating databases for quiz questions and a leaderboard. Initially, we began crafting these databases using MySQL Workbench. After successfully establishing a connection between PyCharm and MySQL, we encountered limitations - limited to local connectivity and requiring identical login credentials across devices. Recognising the need for a more flexible solution, we explored alternatives and found that SQLite offered a viable route for database connectivity.

Although APIs weren't essential for our game, we aimed for a high level of challenge - one that would challenge all skills we have developed throughout the course. Our primary challenge with APIs was reliability on external providers and connections. During the development of the Satellite Challenge, we encountered a recurring issue - our API key was being rejected every few days. Recognising this as a security feature of the API host, we changed our approach: instead of calling the API in real-time and downloading the most recent satellite images, we only called the API during development and saved these images to our code files, thereby hardcoding a simulated API response so we could still use real satellite images without encountering the key rejection issue.

Methods for PyGame for Implementation:

1. **Referencing External Logic** - We employed external logic references for missions including Asteroid Challenge, Satellite Imaging and the Quiz.
2. **Hard-coded Integration** - We directly integrated the Capture Mas mission into the main scene file.
3. **Pop-up Window Usage** - For missions such as the Payload Mission, Asteroid Blast, and Locate ISS, we made use of pop-up windows. This involved running these missions as sub-processes within a new PyGame window.

Agile development:

Our implementation process closely adhered to agile development principles, enabling us to continuously refine and enhance the project. Iterative development empowered us to work in smaller cycles, with regular feedback loops in our daily standups, and prioritise features based on importance. This approach ensured that even with time constraints, we could deliver a polished product while having room to incorporate additional missions and challenges as well as design aspects that we have successfully achieved.

Testing and Evaluation

Testing Strategy:

Our testing strategy was designed to ensure the robustness, functionality, and user-friendliness of *Stratobus Mission*. We adopted a multi-faceted approach that encompassed various testing techniques that we had learned during our Code First Girls degree. We thoroughly debugged our code through the use of "print" statements and in-built debuggers, and used peer review to help identify logic errors.

Unit Testing:

We followed the Test-Driven Design principle and planned out our unit tests before writing any code. This helped our code remain concise and modular. However, we had to adapt our testing strategy during the code-writing process as we learned that PyGame-friendly code needs to be far more modular than the style of coding we were used to. This resulted in far more functions than expected, though thanks to the more simplistic nature of these functions (such as simply outputting a piece of text), unit testing could in some cases be binary rather than testing for valid, invalid or edge cases as we did with more complex functions. The test files are situated alongside each mission script in their individual folder.

Functional and user testing:

Once our game was compiled, we thoroughly tested its functionalities and challenges, and enlisted friends and family to do the same, utilising GitHub's *Issues* feature to track bugs and assign bug fix tasks. Having individuals external to the project was hugely beneficial to provide an outsider perspective - as well as allowing us to demonstrate to others how much we had learned through the Code First Girls programme.

Future Work

- An overall scoreboard - to drive users to excel across all missions.
- Satellite image printing/social sharing - to better bridge the user's in-game experience with their learning (aligning with our core focus on education and engagement).
- Certificates for achievements - to award players' dedication to learning and exploring as well as allowing them to cherish their in-game achievements.
- Additional side-scrolling platform - to further enhance the game experience.
- Multiplayer opportunities with local co-op - to improve the gameplay experience by making it more collaborative and encouraging friends and/or family members to participate together or to be able to introduce the gameplay in a classroom setting (aligning with our core focus on education and engagement).
- Accessibility options - introducing features such as adjustable font sizes and colours as well as audio cues to ensure that players of varying abilities can enjoy the game to the fullest.
- Refactoring scene files - to refactor these into smaller mission-specific files to improve organisation.
- Circular reference resolution - to overcome circular references that occur when attempting to split the scenes file into smaller mission files.

Original Objectives:	**Result:**
Third party APIs & custom SQL database with connector	✅ We have successfully utilised three distinct types of APIs, demonstrating our drive and adaptability through the implementation of different methods (outlined in Methods for PyGame Implementation).
Event listeners	✅ We have used these for many functions including hover functions, sound effects, and responsive key controls.
Database integration	✅ We have used an SQL database to store quiz questions in order to improve security and maintainability. Additionally, we have used a database for the leaderboard query.
Graphics and Audio	✅ We have used an array of different space-themed sound effects for the different missions as well as an ongoing space-themed music which is played throughout the program being run.
Tasks for user to complete	✅ Our original objective was to create four missions where users are able to participate including using a wide range of keys etc. We are thrilled with our finished product as we have exceeded our original plan and have created seven missions all with a good balance of user input.

Conclusion

In conclusion, we successfully achieved all of our baseline objectives and even met some of our stretch targets, such as incorporating additional missions and features like buttons to return to the home screen after completing each mission, resulting in a highly successful project outcome that we are all proud of. Teamwork and accountability were guiding principles throughout the project, and we managed time constraints through careful prioritisation and a modular approach. The diverse range of technologies used in the project contribute to the impact and effectiveness of the finished product.

During our journey, we aspired to implement a global SQL scoring system, allowing players worldwide to compete and compare their achievements. Additionally, we envisioned the possibility of exporting the game as a .exe file for convenient and broader accessibility. The desire to enhance user engagement led us to contemplate introducing levels within the Payload and Asteroid Blast games, complemented by the inclusion of scoreboards to add a competitive aspect making it more engaging. Additionally, we identified early on that a future development of the project could incorporate some hardware into the *Stratobus Mission*, for example Raspberry Pi motion sensors to have the player physically pilot the Stratobus through space, dodging asteroids or other obstacles. We were resigned to the fact that hardware would not feature in our final project, particularly due to the remote element of group members and examiners which would prove a blocker to thorough testing and end use. This led to the innovative incorporation of the Asteroid Blast mission, where players navigate the Stratobus using arrow keys - an inventive compromise that underscores our adaptability and creativity.

Ultimately, the *Stratobus Mission* shows our dedication, flexibility, and curiosity to explore new areas. As we finish this project phase, we are excited to watch our creation keep developing and inspiring, thanks to our innovative thinking and desire to overcome challenges.