Project Report

Introduction

The primary aim of this project is to create an educational game that engages children and challenges their reflexes while encouraging them to learn more about space, ultimately sparking an early interest in STEM subjects. As a team, we all have a shared passion for changing the gender gap in the technology industry, as well as a common interest in science (we are majority sponsored by DSTL, an organisation at the forefront of scientific advancement). We therefore hope the game will inspire young girls in particular to believe that a career in the aerospace industry or any other scientific field is possible; factual content in the game was chosen with this aim in mind, championing the achievements of women in the field. This is a mission we also felt went hand-in-hand with the Code First Girls ethos.

This game also reflects the current public interest in space exploration, with upcoming milestones such as NASA's Artemis III mission - where the first woman and person of colour will be part of a crew landing on the moon - due to take place in the next couple of years.

Roadmap

The **Background and Game Logic** section will highlight the project's context, detailing the design, rules, and logic behind the game.

The **Specifications and Design** section will discuss the differences between functional and non-functional requirements and present the project's architecture.

The **Implementation and Execution** section covers development and implementation (including notable achievements and challenges), team roles, tools and libraries used.

The **Testing and Evaluation** section details the testing strategy, results from functional unit testing and manual user testing, and system limitations.

Finally, the **Conclusion** addresses the project outcomes, reflecting on its potential impact and opportunities for future enhancements.

Background and Game Logic

The premise of the game centres on an astronaut required to dodge asteroids in order to navigate (or 'fly') through space successfully; 1 of 3 lives is lost each time a collision occurs, and the game ends when lives reach 0. The player's multi-tasking ability is also challenged by the display of space-related facts. High scores are recorded and displayed.

Specifications and Design

Functional Requirements

1. Game Start
    - A Welcome Screen should be displayed when the game begins, enabling player name input; gameplay should begin when the enter key is pressed

Project Report

2. Sprite Mechanics
    - The astronaut movement should correspond in real time to player key presses (up, right, left and down)
    - Asteroids should ‘fall’ vertically at varying speed, spawning every 0.5 seconds
    - Facts should slide from the right of the game window to the left, at a slow speed to enable longer reading time, with new facts every 30 seconds

3. Scoring
    - The total lives remaining and health bar should be displayed and updated (lives decreased by 1, health bar decreased by a third) after collisions
    - Gameplay duration and longest gameplay record should be displayed

4. Game End
    - A Game Over screen should be displayed at 0 lives/health, with a personalised message “[Player] the astronaut survived for [duration]”
    - Gameplay should begin again when the enter key is pressed by the player
    - The game loop should stop when the game window is closed

5. Sound Effects
    - The game should have sound effects that correspond to astronaut movements, collisions, game end and a backing track throughout

Non-Functional Requirements

1. Game Speed and Performance
    - The game should start within a few seconds of the code being run, with a consistent frame rate that makes movement slow enough to see clearly, but fast enough to not cause player frustration
    - Both should be achieved regardless of player operating system/computer
    - The game should not crash and quit unexpectedly

2. Game Controls
    - The controls should be simple and accessible for players of all abilities

3. Future Enhancements
    - The game should be easily adapted to increase difficulty (for example, further levels with more/faster asteroids)
    - The game should be easily adapted for use in other subjects

4. Visual Design
    - The game should be visually appealing with attractive, clear background and sprite imagery that makes gameplay easy to follow and engaging
    - The design should have a retro theme as a nod to the history of space exploration and famous space-centred games

5. Game Content
    - The educational content should be age appropriate and all facts checked
    - The game should be interesting and enjoyable

Design and Architecture

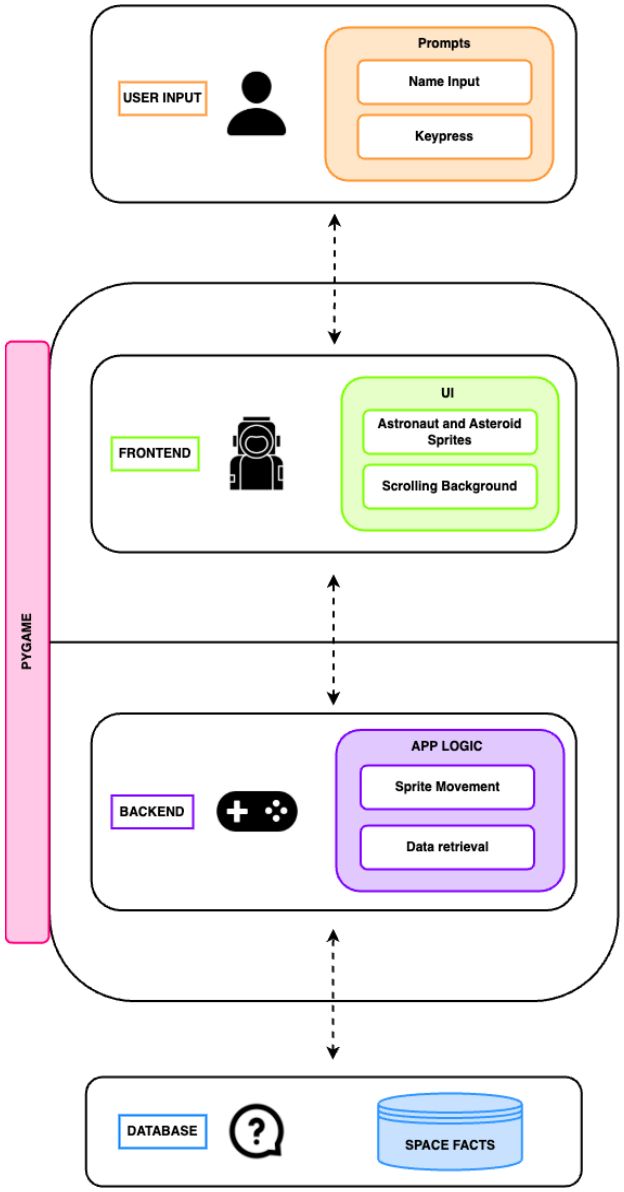


Figure 2: Architecture diagram of the system components for the game

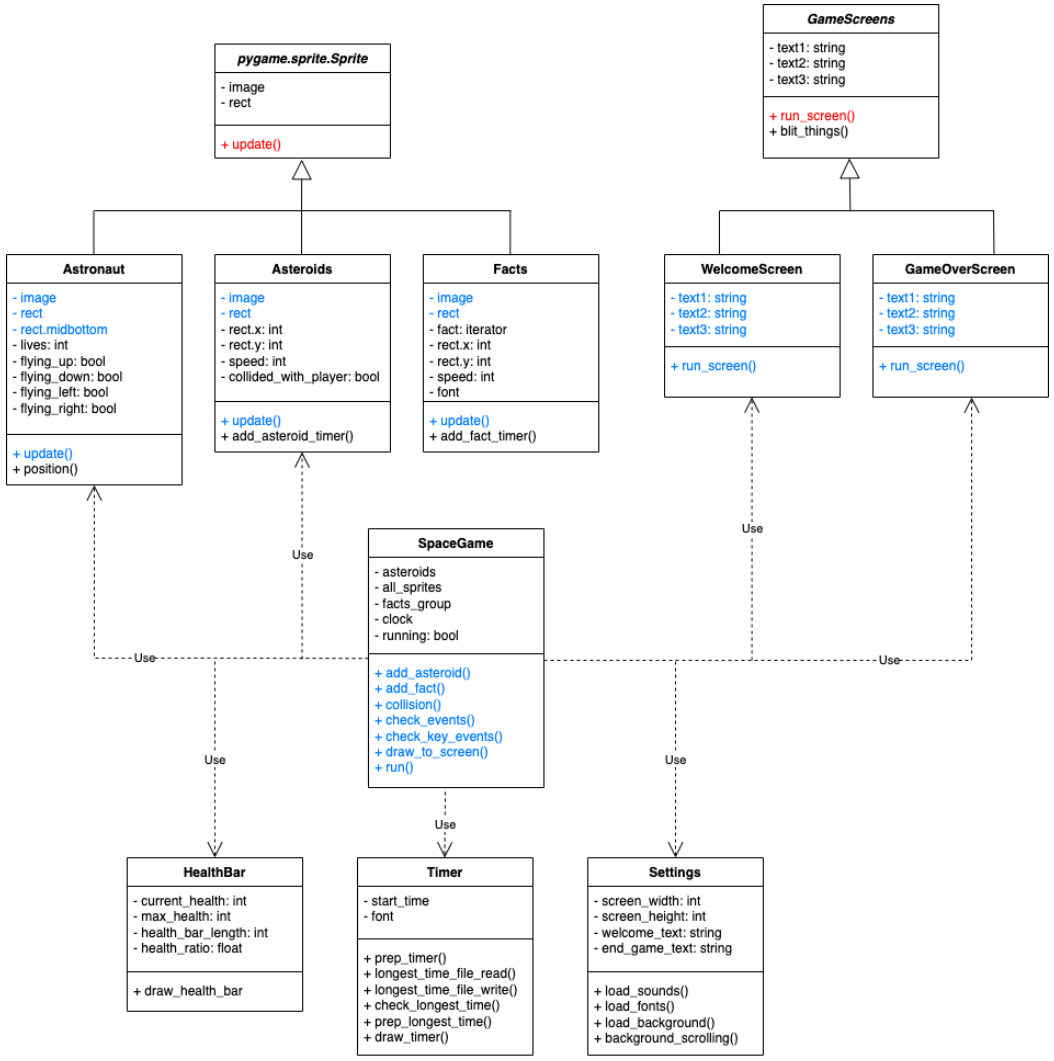


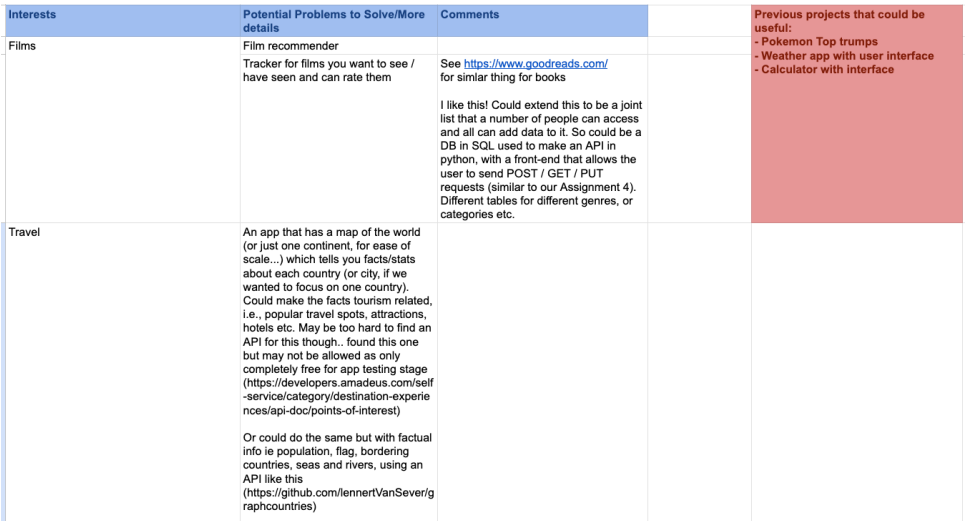
Figure 1: A UML diagram of the class structure for the game, Astro Academy. Database connection and unit test files not included.

Project Report

Implementation and Execution

Development Approach and Implementation Process

Week	Tasks
1	A Slack poll for the top six ideas (all games) from a shared Google sheet (Figure 3) selected a space-themed educational game. We made a Canva board, allowing everyone to add ideas. Further Slack polls were created for divisive design elements (e.g., trivia questions v.s. facts); this overcame the challenge of ensuring all voices were heard. Fundamental game logic was finalised.
2	A timer, facts, and lives were selected as priority components; due to time constraints, we decided against implementing additional levels. We also considered using an API to maintain the integrity of a database containing a leaderboard. However, for the same reason, this was changed to fact storage and retrieval via an SQL-python connector only. Moreover, the longest time was saved to a file and displayed as a more applicable alternative to a leaderboard (single-player game).
3	Python and SQL coding tasks were divided amongst the group according to preference and strength. The Python team compiled the class structure of the game (Figure 1), during a mob programming session; we found this technique extremely helpful when tackling this initial challenge, particularly working together to fix errors. The SQL team created the database and connector code; they also started documentation. Keeping all task delegations and progress up to date on Jira, voting on design amendments in Slack polls, and Sprint Reviews allowed for clear, universal communication. Team calls were also recorded to account for absences.
4	The facts class was implemented last, following successful integration testing of the database and python code (a particular achievement, applying skills gained from lessons and individual research to collaboratively build the game). Initially, while the facts were successfully retrieved, they were only partially visible and moved too fast. After troubleshooting, the code was refactored, and facts displayed while the program paused mid-run. However, this method caused the game to crash on manual testing. As a result, the fact class was refactored for facts to appear during gameplay. Visual and audio features were enhanced with welcome/game over screens, and sound effects.
5	As the submission date approached, we did face technical issues using GitHub, primarily due to merge conflicts when attempting to merge the main branch into our individual branches. In spite of this, we were able to see the benefit of branching, outweighing the additional time taken. Unit testing and manual testing were completed. All team members reviewed the final code.



Interests	Potential Problems to Solve/More details	Comments		Previous projects that could be useful: - Pokemon Top trumps - Weather app with user interface - Calculator with interface
Films	Film recommender			
	Tracker for films you want to see / have seen and can rate them	See https://www.goodreads.com/ for simlar thing for books I like this! Could extend this to be a joint list that a number of people can access and all can add data to it. So could be a DB in SQL used to make an API in python, with a front-end that allows the user to send POST / GET / PUT requests (similar to our Assignment 4). Different tables for different genres, or categories etc.		
Travel	An app that has a map of the world (or just one continent, for ease of scale...) which tells you facts/stats about each country (or city, if we wanted to focus on one country). Could make the facts tourism related, i.e., popular travel spots, attractions, hotels etc. May be too hard to find an API for this though.. found this one but may not be allowed as only completely free for app testing stage (https://developers.amadeus.com/self-service/category/destination-experiences/api-doc/points-of-interest) Or could do the same but with factual info ie population, flag, bordering countries, seas and rivers, using an API like this (https://github.com/lennertVanSever/graphcountries)			

Figure 3: Shared Google sheet for potential ideas/problems to solve ahead of choosing our project objective

Project Report

Team Member Roles

Responsibilities	**Team Member**						
	CN	**EP**	**FC**	**ML**	**MB**	**PC**	**SP**
Game Framework Research	X	X	X	X	X	X	X
Meeting Agendas and Links	X						X
Team Meetings/Sprint Reviews	X	X	X	X	X	X	X
Jira Board Manager		X			X		
Project Assignment	X	X	X	X	X	X	X
SQL and SQL-python connector	X		X	X			
Python	X	X			X	X	X
README.md				X			X
Project Report		X	X	X		X	
Project Presentation	X		X	X			
Unit Testing		X					
Manual User Testing					X	X	X
Code Review		X			X		

Figure 4: Table shows the responsibilities of each team member

Tools and Libraries

- **Slack**: main form of contact between group members, organising meetings, creating agendas and polls, sharing updates on code progress, asking queries
- **Google Drive:** Compiling and organising all shared files
- **Pygame**: game framework
- **Python**: core programming language
- **SQL Connector and MySQL**: to store and retrieve facts.
- **Jira**: to manage project backlog and plan/review sprints (see Figure 5)
- **Canva**: for collaboration, and initial concept design/plan
- **Unittest**: for unit tests of functions within the timer and health bar classes, as well as sprite movement
- **Random**: to generate values for sprite speed, and shuffle fact dataset
- **Itertools:** to access one fact at a time, on an infinite cycle (to avoid errors)
- **ABC:** to build an abstract base class for the Welcome and Game Over screens to inherit from

Project Report

Agile Development

We decided that an agile development methodology was most applicable to our project requirements, enabling us to make the game iteratively and change initial designs when required. This approach also ensured that all team members contributed equally, working on different components at the same time.

We used the following agile elements:

- **Jira:** We created a backlog of tasks and epics at the beginning of the implementation phase, separated into 3 sprints. Tasks were spread evenly among the team.
- **Sprints**: Tasks were divided into short, focused 1-week sprints, allowing us to prioritise deliverables, ending with a Sprint Review to discuss progress and blockers
- **Refactoring**: Code written by one team member was reviewed by at least one or two others, and optimised if needed in order to improve performance and maintainability
- **Pair/Mob Programming:** We utilised this technique in our initial python and database setups, as well as the final code review to improve efficiency
- **Iterative approach**: Features were developed incrementally, allowing for continuous improvement (e.g., the lives rendering, then the health bar as an extra visual aid)
- **Kanban:** We implemented visual task boards on Jira with “to do”, “in progress”, and “done” columns to track completion

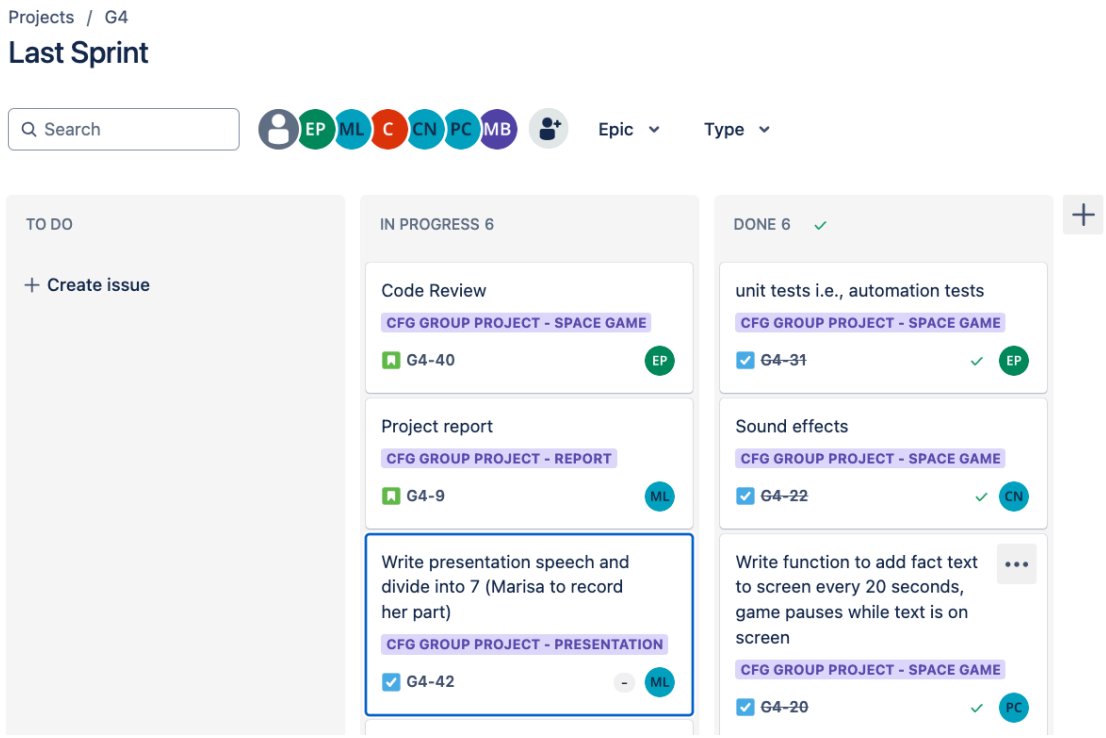


Figure 5: A screenshot of our final sprint on Jira as the project draws to a close, showing task delegation and completion columns

Project Report

Testing And Evaluation

Functional Testing

We utilised unittest in python, testing functions in the Timer class that used logic to return a value: a function to convert milliseconds to minutes, seconds, and milliseconds (a string) and a function to determine the longest duration (strings not converted to reduce code length). Functions were written in a utilities file, so more accessible. We tested edge cases: small/large times; ‘0’ times; and durations with small differences.

The Health Bar class also had a separate utilities file to calculate health bar width. This included division, requiring error handling for data types and zero division. The unit tests tested the game start/end health and ratio values, and error raising.

Finally, we conducted unit tests on astronaut movement. This required a separate file, containing a class that inherited from the Space Game class. The game loop was overwritten with a function simulating a key press and returning the astronaut’s coordinates. Up, right and left keys were tested - not down, as the astronaut initialises at the bottom.

Manual User Testing

The manual user testing is in a separate folder in the repository, with a global README containing a list of each feature tested; links are used to make navigating between the files more efficient. Each feature test was contained in its own file, composed of a README (where the tests were written) and a file for screenshots linked to the tests to give further context. The testing was conducted using Gherkin Syntax: creating scenarios that the user could encounter while playing, and using ‘Given, When, And, Then’ statements to structure a concise description of the test steps and provide the expected outcome of scenarios.

System Limitations

Since the game is designed around space and pygame lacks a built-in physics function, it was hard to find a way to simulate zero gravity. Additionally, pygame is primarily designed for 2D games; if we had any future plans to improve the graphics of the game (such as having asteroids appearing to fly ‘towards the player’), another 3D framework would be needed.

Conclusion

We feel we have achieved our goal of building a fun, engaging game to encourage interest in space and STEM subjects for 7-10 year olds. Given more time, it would have been great to test the game amongst our target demographic, and incorporate further improvements based on this; this could have also been a useful research step in our initial development.

In terms of future enhancements, in order to emphasise the educational element and make the game more interactive, we would like to add multiple choice questions. For the functionality, we would like to explore animating sprite movement, for example, having different sprite images that are cycled through from frame to frame, better emulating the astronaut floating through space, or a flaming asteroid.