­

OCR GCE A  
COMPUTER SCIENCE  
PROJECT

H446-03

Name: Abdullah Noman

Candidate Number: <NNNN>

Institution Name: Salford City College Pendleton

Title of Project: Rocket Agency

TABLE OF CONTENTS

[SECTION A: ANALYSIS 4](#_Toc125548745)

[Project Description 4](#_Toc125548746)

[Overview 4](#_Toc125548747)

[Main Features 4](#_Toc125548748)

[Stakeholders 5](#_Toc125548749)

[Bartlomiej Wierzba 5](#_Toc125548750)

[Osas Osaghe 5](#_Toc125548751)

[Justification of Features and Effects 6](#_Toc125548752)

[Abstraction and Visualisation 6](#_Toc125548753)

[Thinking Ahead 6](#_Toc125548754)

[Thinking Logically 7](#_Toc125548755)

[Research 8](#_Toc125548756)

[Kerbal Space Program 8](#_Toc125548757)

[Space Agency 9](#_Toc125548758)

[Evaluation 9](#_Toc125548759)

[Features 10](#_Toc125548760)

[Essential Features 10](#_Toc125548761)

[Menu screens 10](#_Toc125548762)

[Hardware and software requirements 11](#_Toc125548763)

[Success Criteria 12](#_Toc125548764)

[SECTION B: Design 13](#_Toc125548765)

[Systems Diagram 13](#_Toc125548766)

[Summary of process 14](#_Toc125548767)

[Key variables and data structures 14](#_Toc125548768)

[development test data 14](#_Toc125548769)

[Algorithms 14](#_Toc125548770)

[post development test data 14](#_Toc125548771)

[Sign off proposal 14](#_Toc125548772)

[SECTION C: Developing the coded solution (“The development story”) 14](#_Toc125548773)

[Section D: Evaluation 14](#_Toc125548774)

[Evaluation-final solution 14](#_Toc125548775)

[Project Appendixes 14](#_Toc125548776)

9

# SECTION A: ANALYSIS

## Project Description

### Overview

The problem I am solving is the creation of a space simulation game, where players can build and launch rockets into space and explore different planets. The key features of the game include the ability to build and customize rockets, complete missions while conserving fuel, and explore different planets and celestial bodies. The target audience for this game is ages 10 and up, as the gameplay can get complex as players progress and unlock more powerful parts. The game will be set in a physical sandbox, allowing players to explore and experiment with different gameplay elements.

### Main Features

The main features of the game include:

* Building and customizing rockets: Players will be able to design and build their own rockets using a variety of different parts and components. They will be able to customize the appearance and performance of their rockets, and experiment with different configurations to see how they affect gameplay.
* Completing missions: The game will include a variety of missions that players can complete, such as taking pictures of different planets, landing on a moon, or deploying scientific instruments around solar systems. These missions will require players to carefully manage their fuel and resources and make strategic decisions about their movements and actions.
* Exploring different planets and celestial bodies: Players will be able to explore a variety of different planets and celestial bodies in the game, each with their own unique environments and challenges. They will be able to fly between these different locations and discover new objects and phenomena as they explore.
* Progression: As players complete missions and progress through the game, they will unlock new items and more powerful parts. This will allow them to continue building and customizing their rockets, and tackle more challenging missions and gameplay elements as they become available.

## Stakeholders

### Bartlomiej Wierzba

Bartlomiej Wierzba is a stakeholder in this project. He has a strong interest in the project and is familiar with game development and design through his experience playing games. While he may not have formal training in game development, he has a good understanding of how features should implement and can provide realistic feedback on the mechanics of the game. As a peer, he is also available for frequent communication and can assist with any questions or issues that may arise.

### Osas Osaghe

Osas will be my second stakeholder for the project. He is one of my peers and we frequently talk about software development. Osas is very respectful and serious, and he is ready to give constructive feedback on the mechanics of my project. As a peer, he will be available frequently so I can ask him questions. Although he does not have experience playing games like this one, he has a good understanding of what the game will be like and is interested in my project.

In conclusion, the stakeholders for this project are Bartlomiej Wierzba and Osas Osaghe. Both individuals have a strong interest in the project and are willing to provide constructive criticism to improve it. Bartlomiej Wierzba has a background in game development and frequently plays games, which makes him a valuable stakeholder as he can provide insights on how features should be implemented and offer realistic feedback. Osas Osaghe is a peer who has a good understanding of game development and is available frequently to provide feedback and assistance on the project. Both stakeholders will play a crucial role in the success of the project and will be an integral part of the development process.

## Justification of Features and Effects

### Abstraction and Visualisation

Abstraction is a key aspect in the design of this game, as it helps to create a more user-friendly experience by simplifying the gameplay and making it easier for players to understand. To achieve this, the game will use 2D graphics rather than 3D, which will make the game clearer and require fewer controls for players.

To keep the game mechanics simple and accessible, some real-life physics have been abstracted. For example, the air resistance in the game will have a linear falloff based on altitude, rather than considering high- and low-pressure pockets and resulting wind patterns. Similarly, the gravitational pull of planets is constant across the surface and inversely proportional to distance from the planet's center, rather than considering complex calculations like Kepler's third law and Newton's laws of gravitation.

To keep the gameplay moving at a fast pace, the distances between planets in the game have been shortened. Abstraction also helps improve performance by reducing the number of calculations the game must make per frame. For example, lighting is kept simple and there are no other objects in space, such as asteroids, which could strain the computer. When rockets crash, they simply fall apart, rather than deforming, and an explosion sound and effect is played, rather than simulating a complex physical event. These changes allow the game to run more efficiently on a wide range of devices.

In summary, the use of abstraction and visualization techniques in the design of this game creates a user-friendly and efficient gameplay experience, while still providing a sense of immersion and excitement for players.

### Thinking Ahead

Before a new game is saved, the positions and sizes of planets are randomly generated so that they can be mapped out on a 2D plane. This allows players to see all the available planets on the map, rather than having them randomly generated as they explore space and potentially limiting the number of visible planets.

Progression is a key aspect of the game. Players will start with basic thrusters and fuel storage, which allow them to fly up into the atmosphere but not leave it or create a stable orbit around the first planet. As they complete missions and progress through the game, they will unlock more advanced items and features, such as a science module that can record data at various locations and exchange it for science points to unlock new areas of the solar system or galaxy.

To ensure a smooth and responsive gameplay experience, the game updates the positions of objects in space every frame. It does this by using the velocity attribute of each object and adding it to the current position to calculate the new position. This update occurs before any drawing, which allows the game to display the updates in real time and feel snappier to players. If there are any changes to an object's velocity during a frame, such as a key press altering the velocity, these changes should be made before the new position is calculated for the same benefits.

### Thinking Logically

The main game loop will be responsible for calculating the velocity and movement of the objects in space in this game. To create branching paths in the game, checks will be performed to ensure that the player has unlocked certain parts or is in the correct location to complete certain actions, such as taking a picture. The game will be developed using Pygame, which offers a range of features and functionality that are well-suited to the needs of this project.

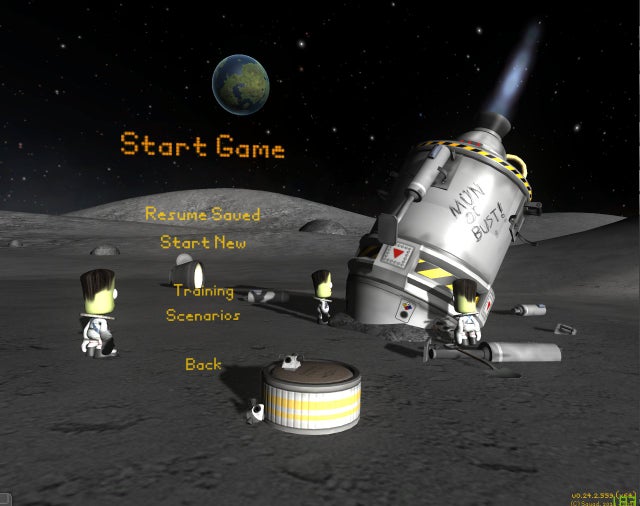
To keep the mechanics consistent and visually appealing, the gravity in the game will be simplified and calculated based on the radius of the planet rather than its mass. This will prevent small planets from having a disproportionately large sphere of influence, which would look unrealistic.

To ensure that the game runs smoothly and efficiently on a variety of devices, certain aspects of the gameplay will be abstracted. For example, lighting will be kept simple and there will be no other objects in space such as asteroids or debris. When rockets crash, they will simply fall apart rather than deform, and an explosion sound and effect will be played, rather than placing strain on the computer by simulating a complex physical event.

By breaking the problem down into smaller, more manageable pieces and using algorithms to solve them, it will be possible to create a dynamic and immersive gameplay experience for players. The structure of the solution will be carefully planned to ensure that it is effective and efficient, and usability features will be incorporated to enhance the player's experience. The key variables, data structures, and classes will be identified, and any necessary validation will be implemented. Test data will be used during the iterative development of the solution, and any further data will be used in the post-development phase to evaluate the success of the solution.

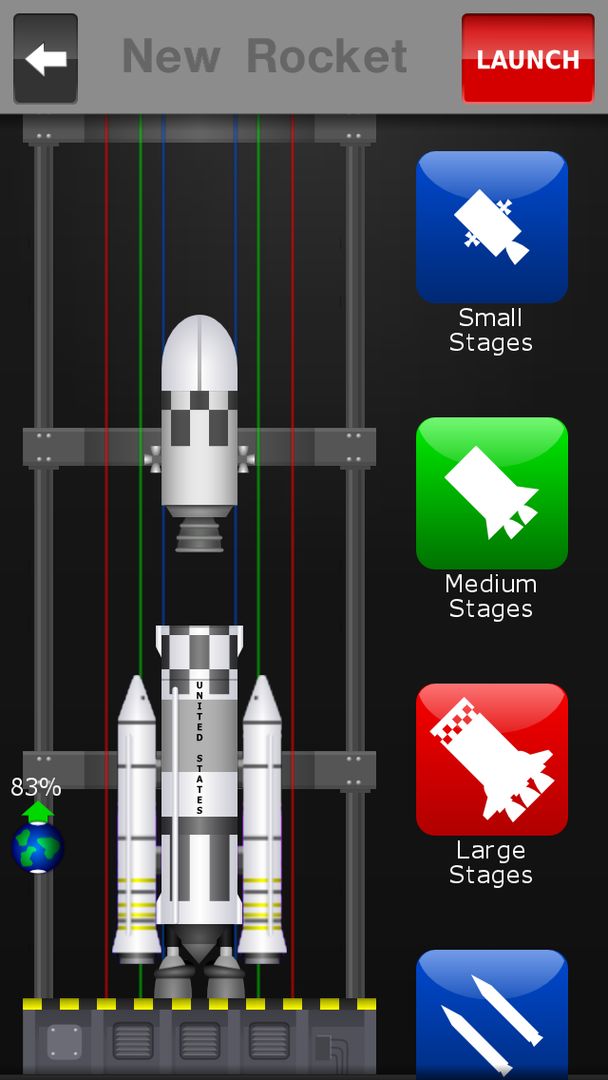
## Research

For my research, I looked at two existing games that are like my own, Kerbal Space Program and Space Agency. My goal was to gather inspiration for features and approaches that could be useful in my own game, which will be developed using Pygame.

Kerbal Spchromeace Program  


Kerbal Space Program is a 3D space simulation game that is quite technical and intricate. It is only available for PC due to the high computational power required. Players can create their own rockets in 3D using a variety of different parts and explore different planets with their rocket. The game has an in-game map that allows players to see all the planets and their ships. It also features a variety of buildings where players can perform different tasks such as hiring crew members, building rockets and planes, managing contracts, and planning strategy.

A screenshot of a video game

Description automatically generatedSpace Agency  


Space Agency is a 2D space flight game that is simpler and available as a mobile game. It will serve as inspiration for the visuals in my own game, although I will need to adapt the layout to fit a wider aspect ratio. The game features a rocket builder building with a variety of parts and subcategories of smaller components. It also has a rocket launch system that displays information such as fuel levels, monopropellant, and launch time. You can build your rockets in a similar, but simpler fashion than in KSP because it is 2D instead.

### Evaluation

Overall, these games are relevant to my own game and offer a variety of features and approaches that I can draw inspiration from. However, my game will have some limitations compared to these more complex games, such as a simpler physics model and a more abstracted approach to rocket building. To ensure that my game is successful, I will need to identify and prioritize measurable success criteria, such as the number of successful launches or the average playtime of users.

## Features

### Essential Features

The game will include key features that are aimed at enhancing the player's experience. These features include interactive menu screens, a rocket building system, a rocket launch system, and a mission control building.

#### Menu screens

The game will feature an intuitive and user-friendly menu screen, complete with a background image and clickable buttons that make it easy for players to navigate the game and adjust settings.

The pause menu will enable players to switch between different buildings and missions, as well as access a tech tree menu where they can select their next advancements.

Additionally, the game will include a save system menu, allowing players to save and load previous progress.

**Justification:**

This is so people can navigate the game and change their settings and go into different screens / progress in the game, save / load old saves.

#### Rocket Builder Building

The game will offer a dynamic and easy-to-use building system that enables players to construct rockets using various parts from a menu.

The system will automatically calculate important rocket parameters such as Delta V (change in velocity) and Thrust to Weight Ratio (TWR) as the player builds their rocket.

Players will be able to use drag-and-drop functionality to attach parts to the rocket, starting with the main stage that offers the best performance at sea level and has a favourable TWR.

As players progress through the game, they will unlock additional parts with a range of capabilities, giving them the ability to build larger and more powerful rockets.

**Justification:**

This is so users can pick the parts they want and allow them to access the progression system and build their rockets.

#### Rocket Launch System

The game will include a launch system that presents players with a Heads-Up Display (HUD) on or around the rocket. The HUD will provide important information such as fuel levels, Delta V, and other rocket performance metrics.

The HUD will also display information about the monopropellant, which is used for making fine adjustments to the rocket's trajectory.

Additionally, the HUD will display arrows on the side of the screen indicating the position of planets relative to the rocket, so players always know their position during their journey.

**Justification:**

This is so players can read their rocket stats and find out where they are relative to other planets. This also allows the user to control the rocket.

#### Mission Control Building

This building will have your rocket and the planets on this screen. This screen is so you can track and set planets as your target, so it shows up more information about the planet, e.g., how far it is, the ETA, etc..

**Justification:**

This is to allow users to see a complete map of their planet system and understand the relation between the planets and their sizes, atmosphere information, and details about planets. This is mainly for quality of life for the user.

#### Rocket information HUD

This is the display the player would see when they are in-flight details such as the amount of fuel they have and plan their journey, this is also the screen where they would see planets and should be at the top of the screen, so it doesn’t obstruct the main view

**Justification:**

This is so the player can see and plan their journey, this allows the user to see where they are going, their velocity, rotation, heading and fuel status all in a concise manner.

## Hardware and software requirements

Python 3.6+ installed, the game will automatically install Pygame if it is not installed.

## Success Criteria

|  |  |
| --- | --- |
| Save / load system | Allows the player to save their progress so they can continue where they left off and have multiple saves. |
| Pause menu | The pause menu allows you to revert a flight back to launch in-case the rocket blows up. & Save and exit the game or save and go to main menu. |
| Building switch system | Ability to go into different menus and disable others seamlessly. |
| Timer | There should be a timer to display how long you have been in a rocket launch. |
| Graphics to show when the rocket is running | The fire below the rocket (plume) |
| Camera shakes when the rocket is going fast in atmosphere | This is to give the user a more immersive experience launching rockets and gives the game a more premium feel. |
| Staging system | The stages should detach as the user wants them to and display a small animation of them moving away from the rocket once detached, this is for when a stage of the rocket runs out of fuel, you can get rid of excess fuel and switch to a more efficient engine that works better at a specific atmospheric pressure. |
|  |  |
|  |  |
|  |  |

# SECTION B: Design

## Systems Diagram

A screenshot of a computer

Description automatically generated with low confidence

## Summary of process

* ­ Save Game:
  + Open a file and save the game data into it, preferably in a JSON file as it has good formatting and is readable for debugging and is like python dictionaries. This format makes it, so it is easier to serialize data.
* Load Game:
  + Open a previously saved game data file and read the JSON, then desterilize the data into the program.
* Main menu:
  + The main menu has the buttons for settings, help, credits, a quick save button and a quit button.
* Building selection:
  + Display buttons for the different buildings so it is easy to swap between them.
  + This includes, tracking station, contract menu, hiring menu, vehicle assembly bay, and the tech tree menu.
* Tech tree menu:
  + Display the currently loaded tech tree in a graphical user interface.
  + On press of button if the user has enough credits unlock that node
  + If node before that isn’t unlocked, do nothing.
* Tracking station menu:
  + Get the array of all the spacecraft that are up right now, display the list & their positions on a map.
  + On click of a spacecraft, let them either self-destruct or control it.
* Contracts menu:
  + This displays a list of quests that you can agree to do, this will out a bounty on a quest.
  + Once a quest completes, money is added to the player’s balance
* Vehicle assembly bay:
  + Display a list of rocket parts, the ones that are not unlocked by the nodes are not displayed as greyed out.
  + Load a rocket from a file.
* Build rocket:
  + When image clicked, put it on the mouse and when it is dropped in the middle, place it down as a stage.
  + Finalize a rocket after the finish button pressed.
  + Save rocket to file.
* Display HUD:
  + Display the fuel, monopropellant, time, etc.
  + Display the quit button & positions on the edge of the screen pointing to planets.
  + Display velocity.
  + Display lines on showing where the rocket is moving.
* Display Planets:
  + Load images from planets folder
  + Draw planets at position if the position is less than the radius away from the edge of the screen.
  + Draw the gravity ring around the planets (their sphere of influence)
  + Rotate the planet image on the screen to make it look like they are rotating.
* Display Vessel:
  + Show on-screen controls of the vessel.
  + Check if vessel is colliding with something else, if it is, destroy the vessel.
  + Recover vessel if it is on the home planet for funds.
* Control vessel:
  + Take keyboard inputs to control the vessel and move the rocket around.
  + Take input to escape into the menu and

## Key variables and data structures

* Display surface (variable: surface)
  + The variable used to store the display buffer on which you can draw shapes and images.
* Rocket speed (variable: self.vel in class Rocket)
  + This is used to store the speed and heading of the vehicle.
* Rocket fuel (variable: self.liquidFuel in class Rocket)

## development test data

* Main menu
  + Button highlights when hovered, check button does call back when clicked.
  + Background image displays.
  + Text renders in the middle of buttons.
* Building menu
  + Buildings display.
  + Clicking a building does call back.
* Rocket builder
  + Dragging rocket parts into the assembly places the part at the bottom, above the last part if applicable.
  + You should not be able to connect two incompatible rocket parts together
* Tracking station
  + All current launched rockets are displayed in the map and are able to be switched to when clicked.
  + All rockets are updating their position on the display every time their position is changed.
  + Rocket names should be displayed next to their icon.
* Rocket launching
  + The rocket should be able to provide an increase in velocity if there is an engine at the bottom.
  + The rocket should not be able to produce thrust if there is no fuel.
  + The different types of movement should use different types of fuel, such as monopropellant for the rotation and the main fuel for the main engine.
* Missions / contracts
  + Contracts can be claimed by pressing the button.
  + Completing the criteria for a contract gives the rewards for completing it.
  + You can not accept more than 5 contracts at a time.
  + Accepting a contract will give funds to achieve / fund the mission.
* Tech Tree
  + You should be able to unlock a part if you have enough science points.
  + You cannot unlock the same part twice.
  + You cannot unlock the part if you do not have the science points to unlock them.
  + You can only unlock parts immediately after the latest unlocked part, meaning you cannot skip a part in the tech tree line.

## Algorithms

## post development test data

## Sign off proposal

# SECTION C: Developing the coded solution (“The development story”)

## Prototype 1

### Objective

A menu with clickable buttons and text inside of them which show what their function is.  
Once a button is clicked the appropriate action should be taken. This would be created as a class called menu that can be toggled on and off. When it is on it should display the menu, otherwise it should not.

#### Step 1: Creating a window

Text

Description automatically generated

#### Explanation

This code should create a window of size 512x512 and draws a 2D gradient pattern to ensure that window is working correctly. It should have a handler for input events and run at 120fps as defined in the constructor.

### Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| #Num | Description | Expected Result | Actual Result | Outcome |
| 1 | Check if window is displayed | It shows up | It shows up | Pass |
| 2 | Check is pattern is drawn | It is drawn | It is drawn | Pass |
| 3 | Check if window is at 120fps | It is getting updates | It is frozen | Fail |

Graphical user interface

Description automatically generated with low confidence

#### Remedial Action

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | Check if window is at 120fps | It is getting updates | It is frozen | Fail |

Shape

Description automatically generated

# Section D: Evaluation

## Evaluation-final solution

# Project Appendixes