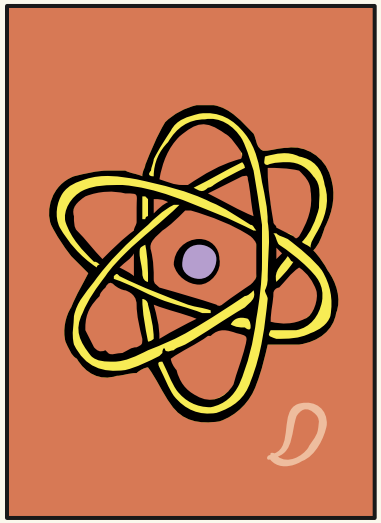
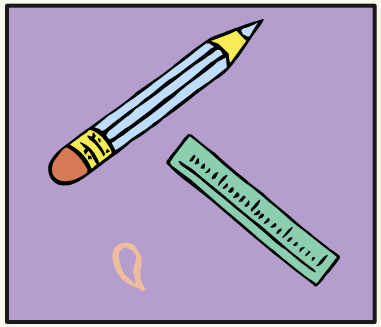
CZ3003

CASTLESKY

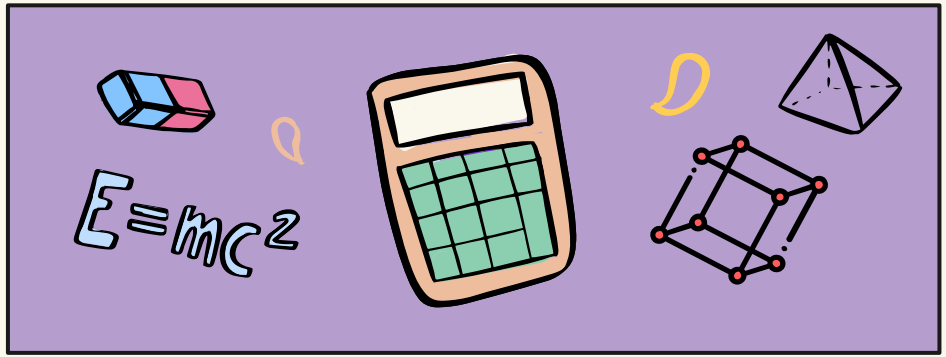


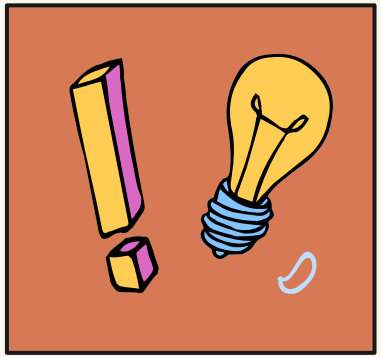


Gamifying and Socializing Teaching and Learning

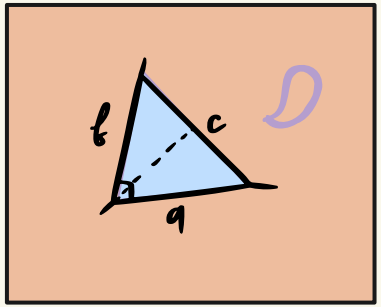
Table of contents

01





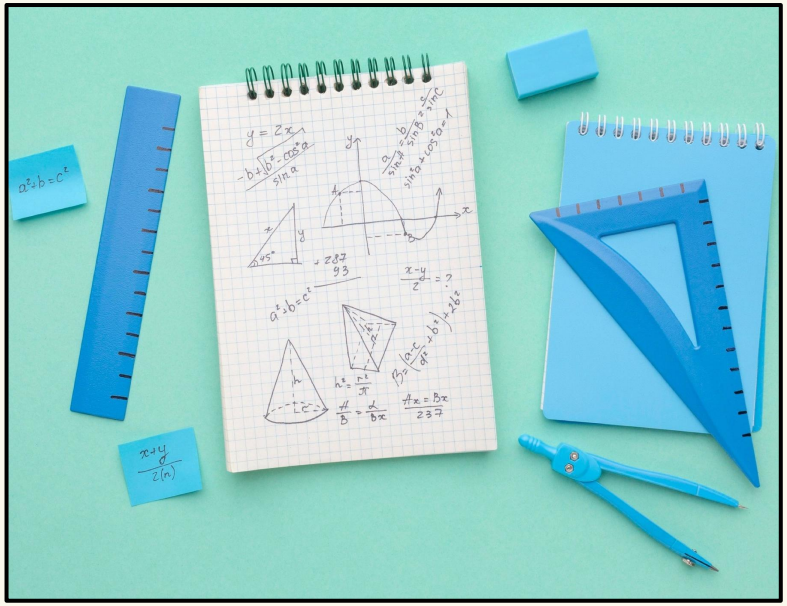
Introduction



Problem Statement, Objectives, Gameflow

CasTheSky

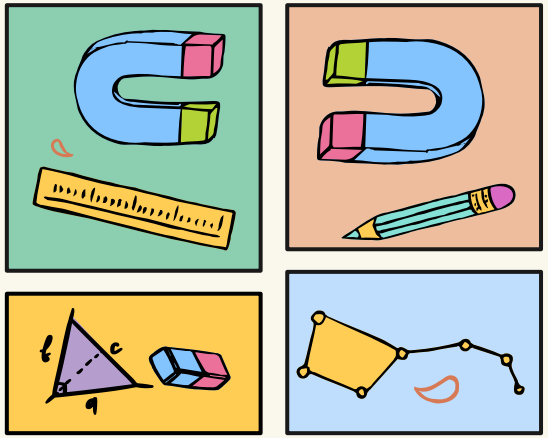
A social application aimed to facilitate learning in the form of gamification for primary schools



Problem

Students

- Usually face the standard mode of education
- Unable to see how what they learn in school can be used in real life



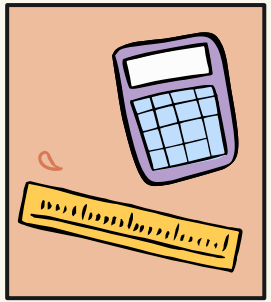
Teachers

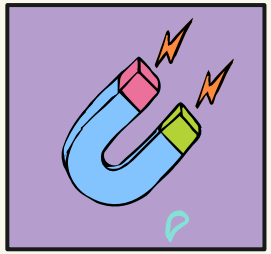
- Struggle to get the attention of students
- Want to inspire students and teach them in more interactive ways

Features

Personalisation

01



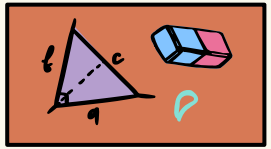


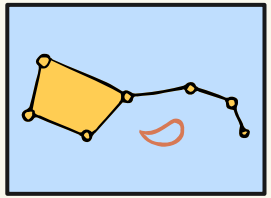
03

Countries

Worlds

02





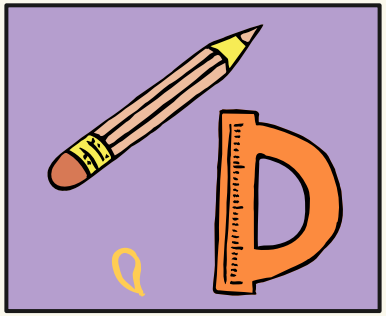
04

Levels

Product Functions - Students

Navigation

Navigation of character to explore a world and across different worlds

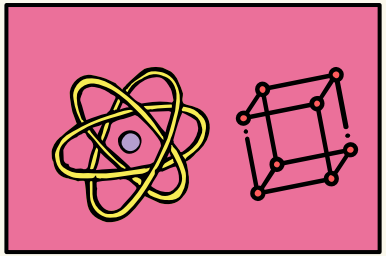


Mini-Games

Playing mini-games that differ in difficulty, last level is re-attemptable

NPCs

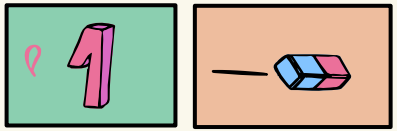
Players are able to interact with NPCs to get tips and tricks on a topic



Leaderboard

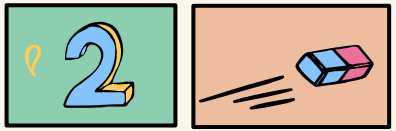
View leaderboard to see scores compared to other players

Product Functions - Teachers



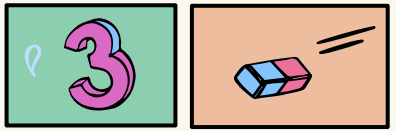
Summary Report

Get a summary of the students' scores in a class in terms of visualisation charts



Generate Access Code

Unique access code for a class to unlock a topic which can be shared on Whatsapp



View Leaderboard

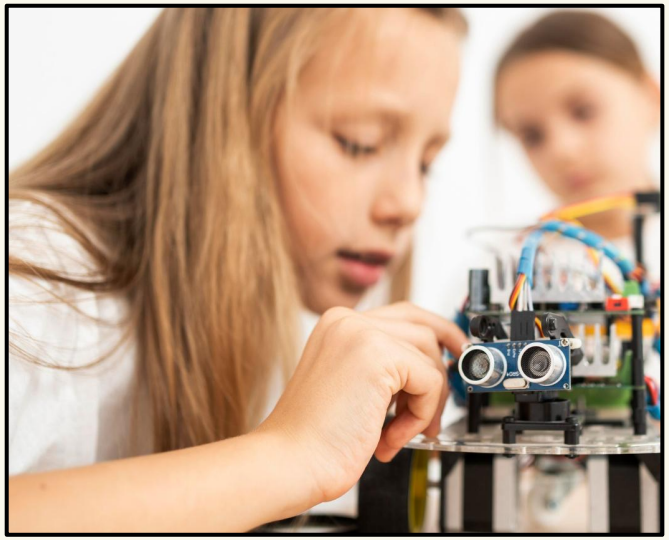
View leaderboard of students in a class based on a distinct world and country

Gameflow

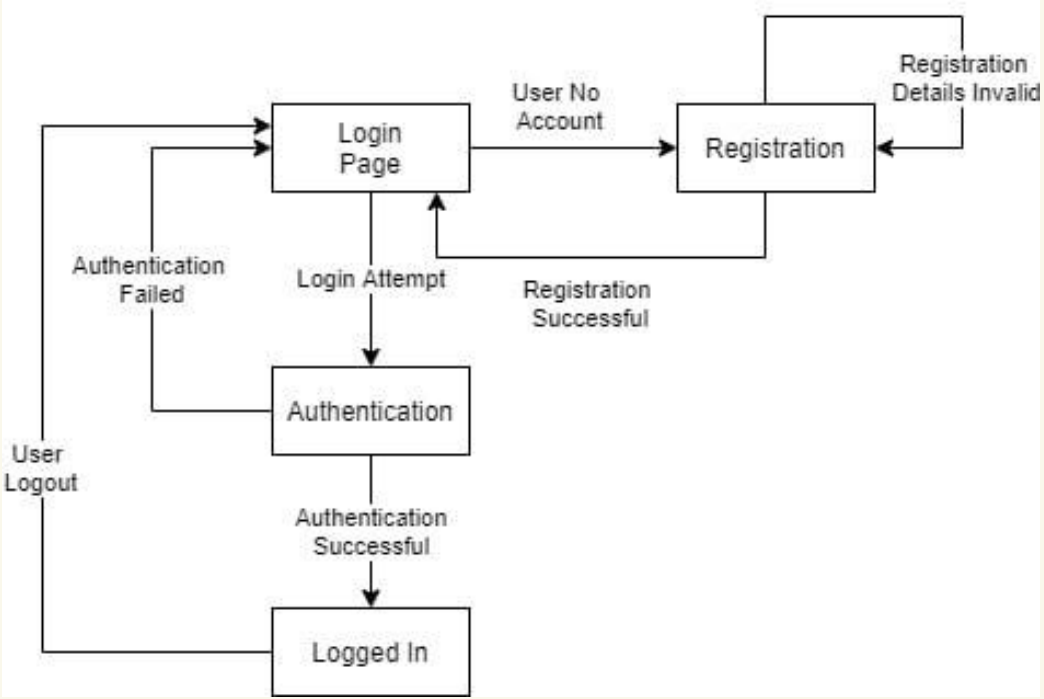
A Authentication

B Student

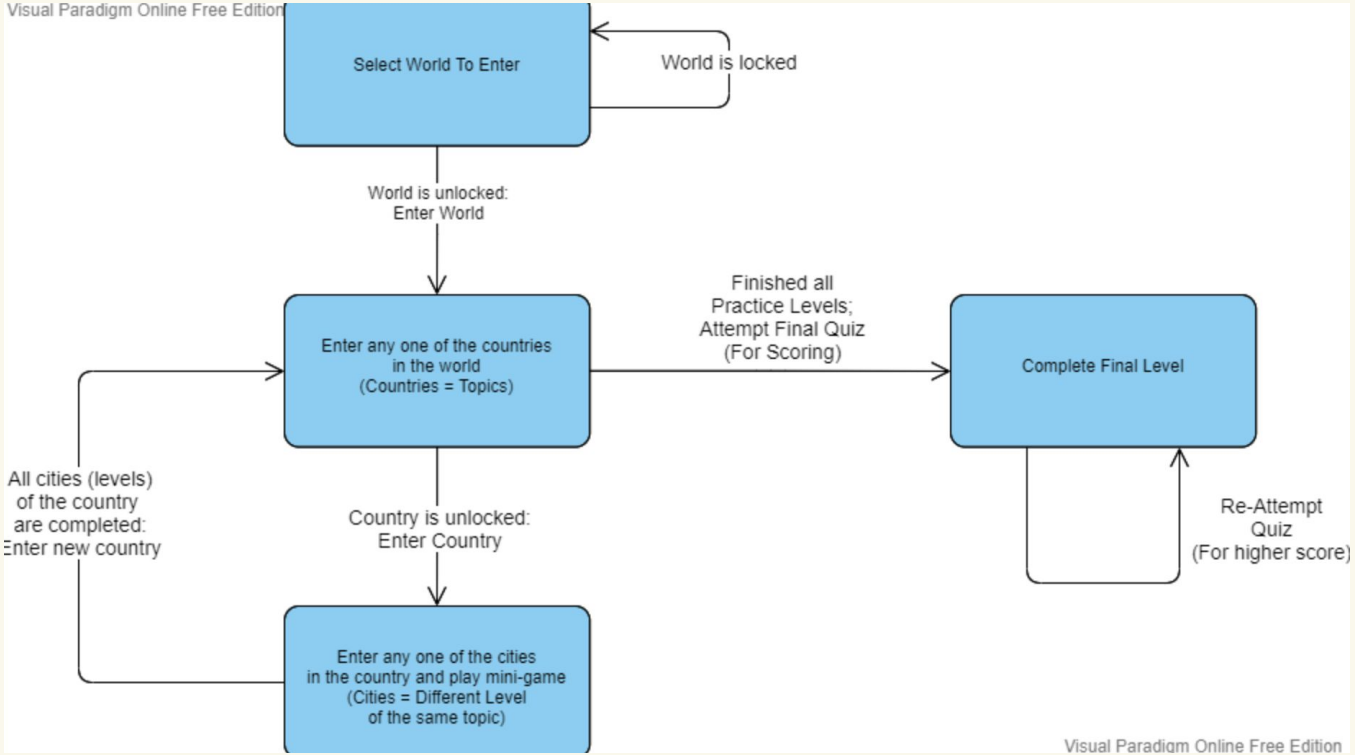
C Teacher



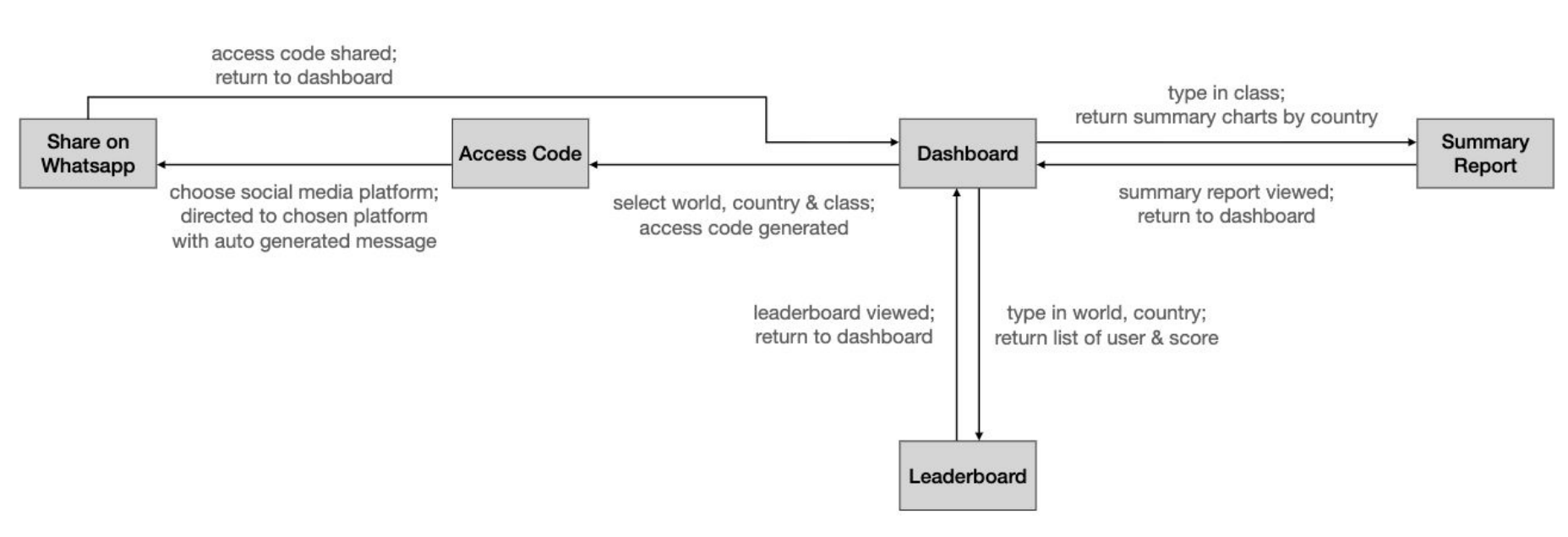
Authentication Route

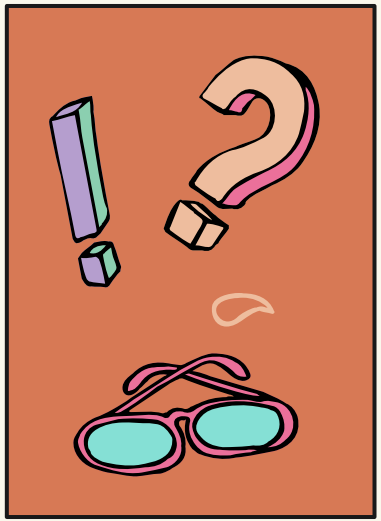


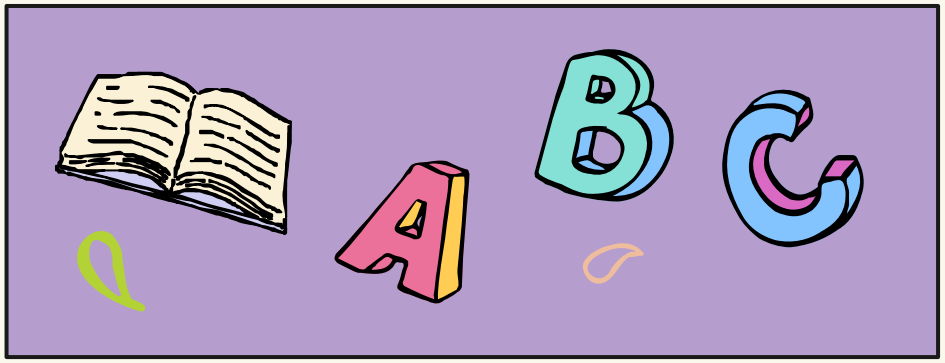
Student Route



Teacher Route

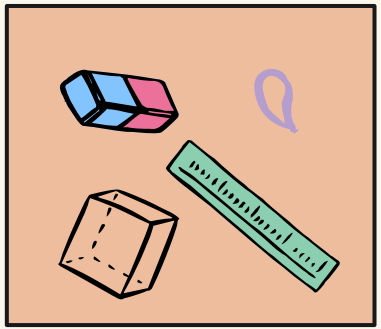




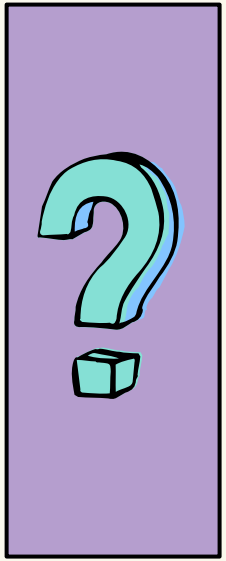


02

Quality Attribute

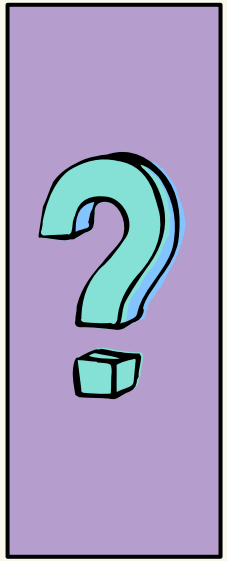


Performance, Flexibility, Maintainability



What is a Quality Attribute?

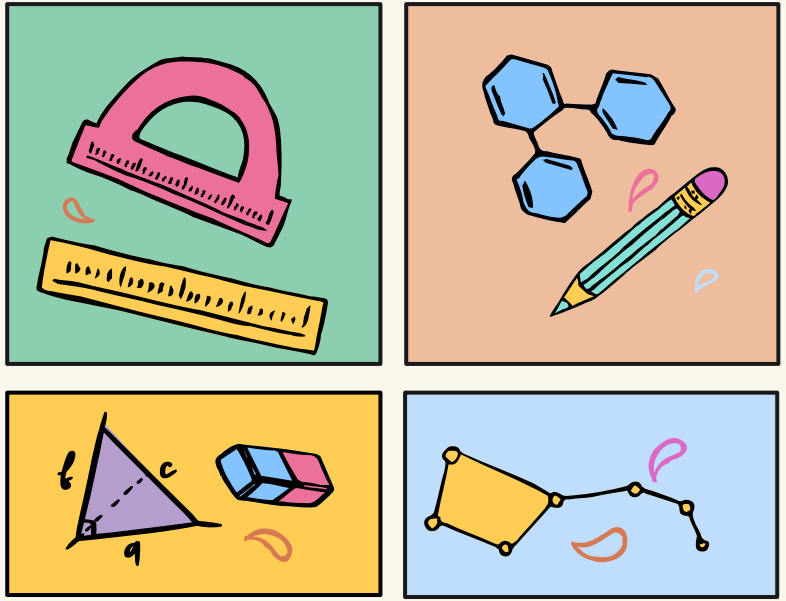
Software Quality Attributes are features that facilitate the measurement of performance of a software product. High scores in these features enable software architects to guarantee that a software will perform as the specifications provided by the client.



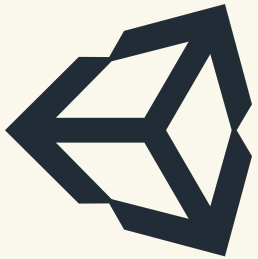
Performance

Performance is an indicator of how well a software system or component meets its requirements for timeliness. Performance was chosen because:

- Target audience are students
- Tend to have low attention span
- Arguably most important



Code



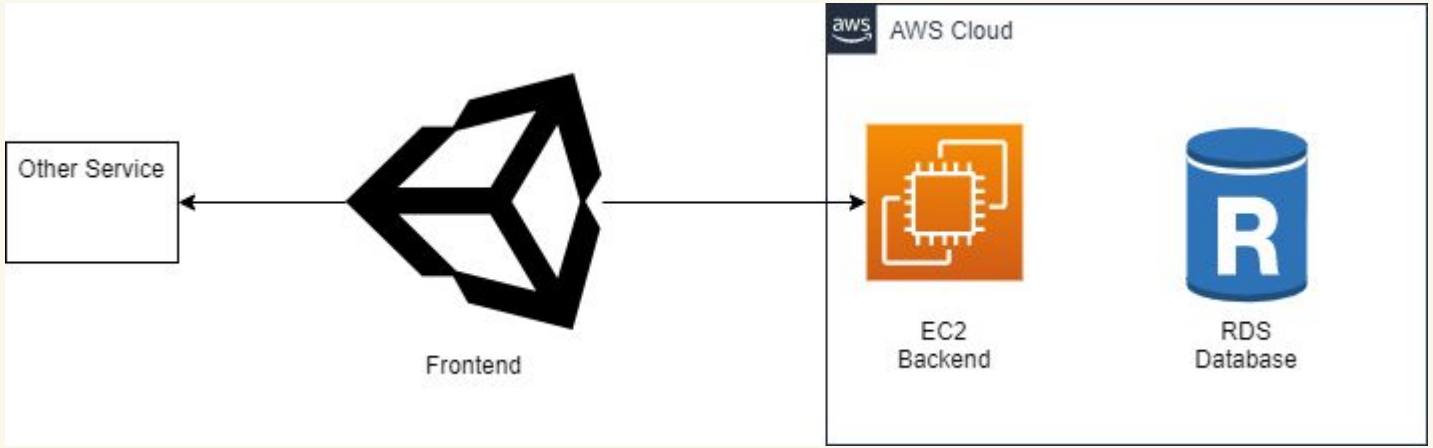
Unity Frontend

- Scripts are developed modularly
- Smaller assets decrease loading time
- Relative performance increases

Nodejs Backend

- Variant of MVC architecture
- Load Minimum Data
- Data transmission time decreases
- Relative performance increases

Architecture



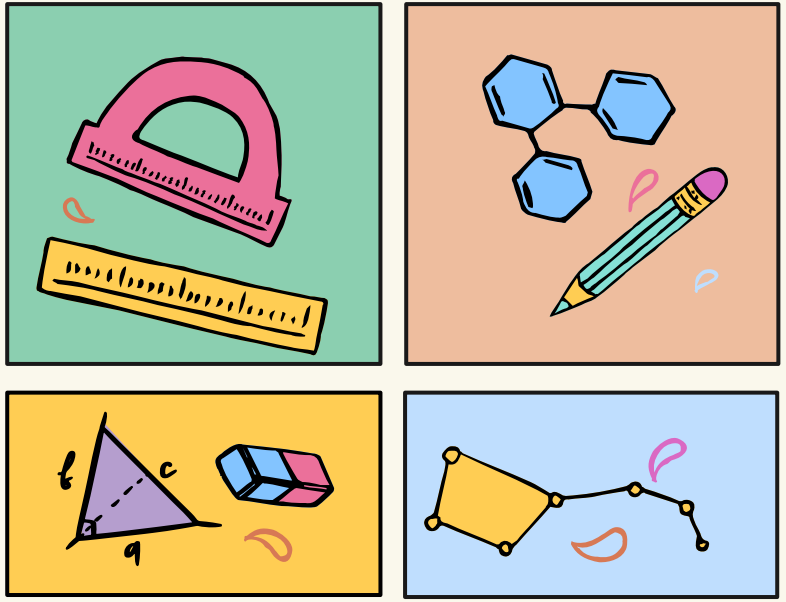
Services/Framework

Flexibility

Flexibility refers to how easy it is to add/modify new capabilities to the software, especially with regard to changes in requirements

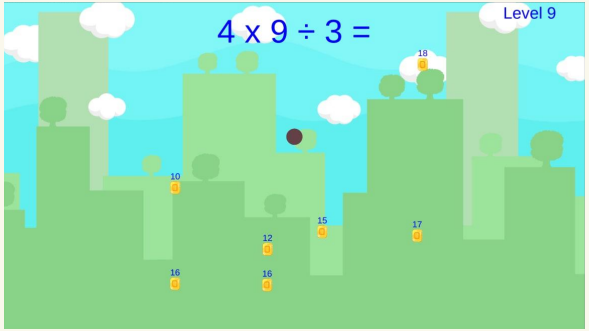
Flexibility was chosen because of

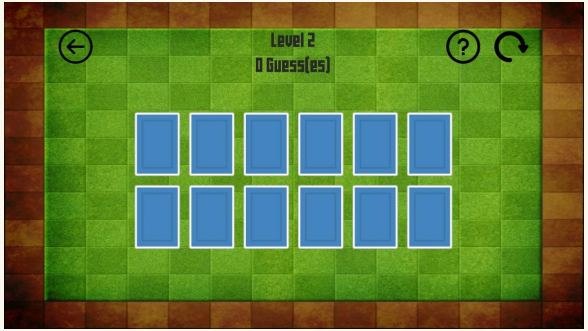
- Likely requirement changes because of changing educational requirements
- Large range of applications across different educational disciplines

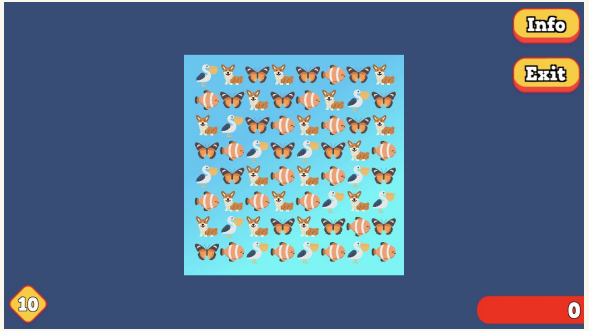


Flexibility

- **Addition/modification of new minigames**
  - Independent development ensures games are easily modified







Flexibility

- **Addition/modification of new minigames**
  - Independent development ensures games are easily modified
- **Addition of new minigame questions**
  - Abstracting the contents of the questions in the minigame from the minigame itself allows for developers to be able to load the minigame containing different sets of questions
- **Addition of new levels and topics**
  - The Minigame Subsystem interface allow for relatively easy duplication of the minigames, and combined with the flexibility of question contents, allows for many variations of the minigame to be implemented in the CastleSky

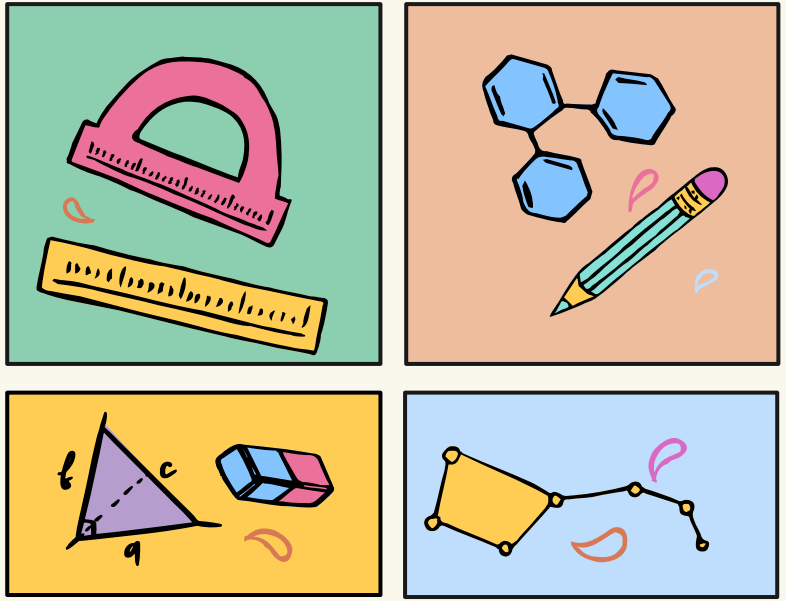
Flexibility

- **Database design and structure**
  - The database is structured for reusability and flexibility
- Database is **highly normalised**
  - Allows for inclusion of additional attributes easily
  - If user requirements call for database updates, the game will not break
- Backend subsystem **highly modular**
  - Following the MVC framework, modification of models and controllers to reflect changing requirements is easily done
- Unity Components used primarily employ Interfaces to communicate with each other, reflecting the Single Responsibility Principle - easily modified to reflect changing requirements

Maintainability

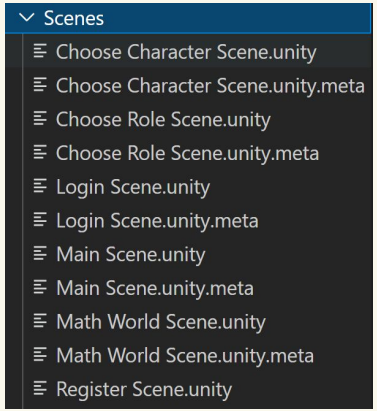
Maintainability refers to the overall readability and structure of the code base, as perceived by other, external developers

- Due to the flexible nature of CastleSky, many developers are likely to be interested to contribute
- Code should be easily maintainable and reasonable to encourage contributions to the repository



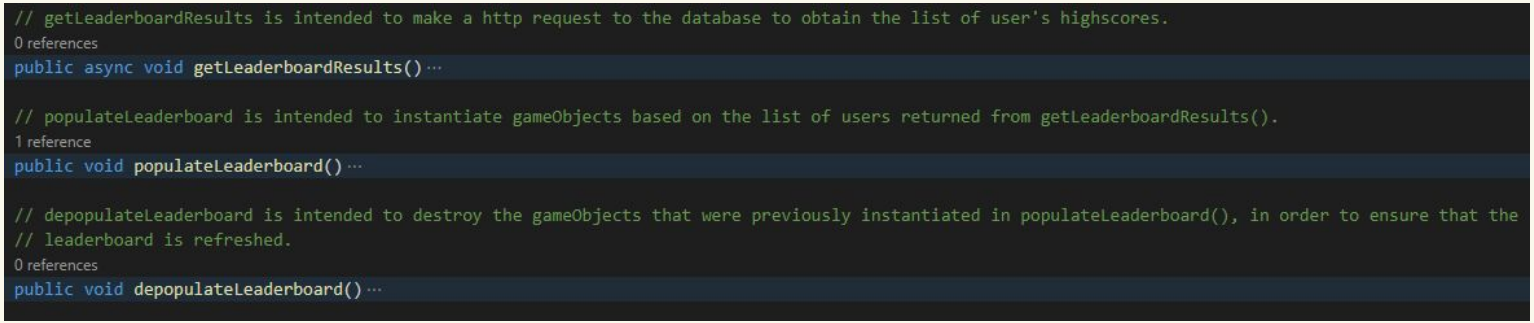
Maintainability

- Highly detailed file naming conventions, which highlights emphasis on readability
  - Developers will be encouraged by understandable file names
  - Experienced developers will be able to jump into development much more quickly



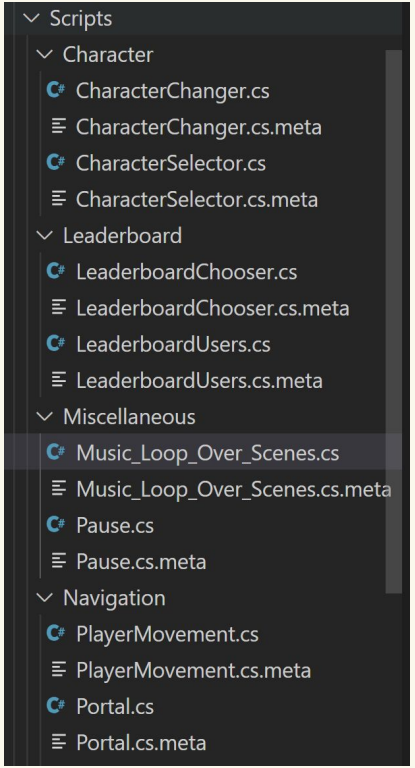
Maintainability

- Detailed code-level documentation is provided for functions that are more complicated
- All methods and functions used are named appropriately
    - Provides a much more readable and maintainable codebase for external developers

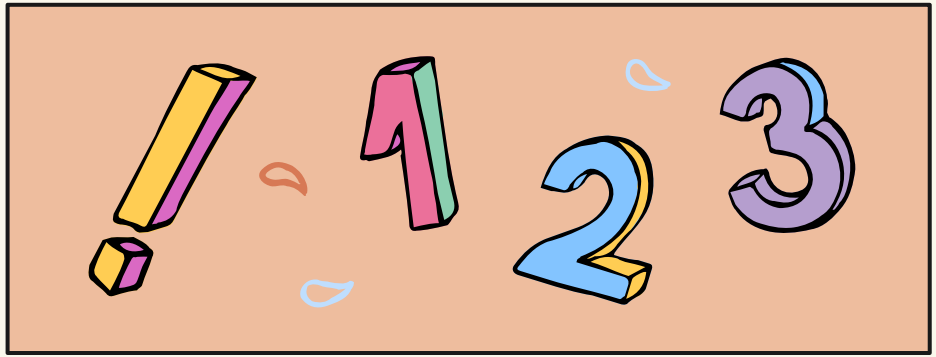


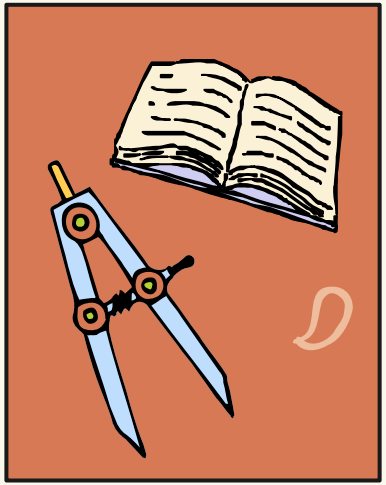
Maintainability

- High level of abstraction for scripts used
  - Various scripts used to execute various functionalities within the game
  - Clear script naming and abstraction into classes makes code much more maintainable

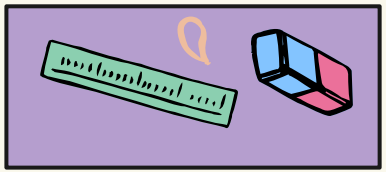


03





Architecture



Defined Subsystems, Architecture Design Choices

Subsystem Design Goals

Designed for **Flexibility**

Improves **Performance**

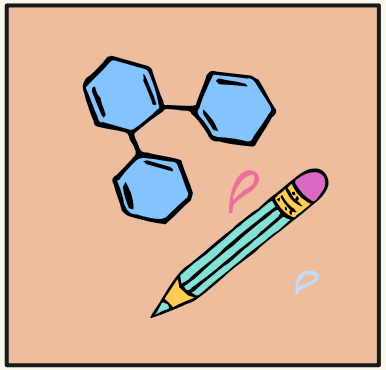
SOLID Design Principles

SRP

DRY

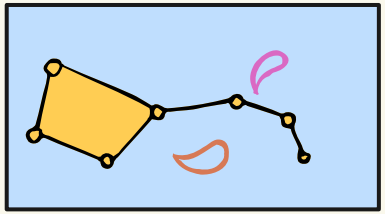
External Subsystem

- Responsible for **interactions with external applications/APIs**
- Modular design allows for **flexibility**
  - Support for other applications in the future
  - Easily understood, **maintainable**
- Implements sharing functionality



Backend Subsystem

- Responsible for communications between game and database
  - node]s backend
    - Developed following the **maintainable** MVC framework
- All communication requests with the database have to go through this subsystem
- Abstracts database operations into more **maintainable** classes
  - **Scoring Interface**
    - CRUD for posting minigame scores
  - **Authentication Interface**
    - Register and Login operations
  - **Minigame Data Interface**
    - Required data for loading minigames

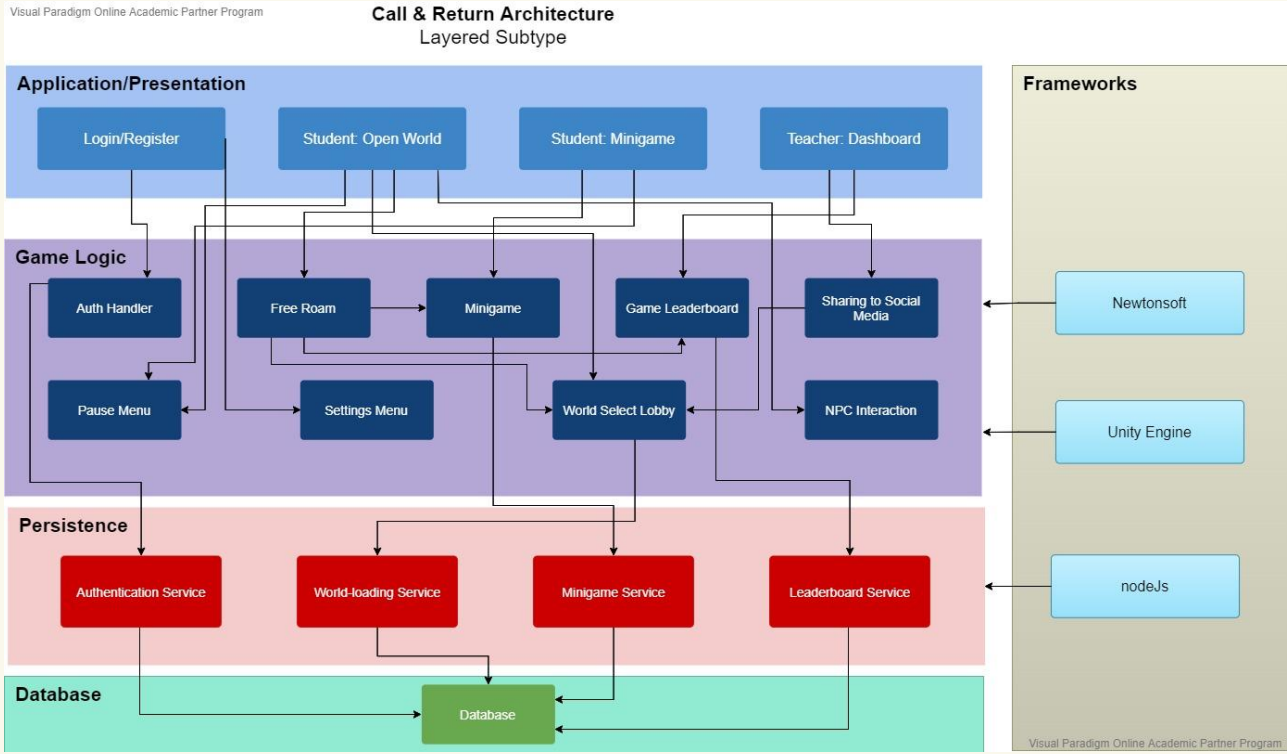


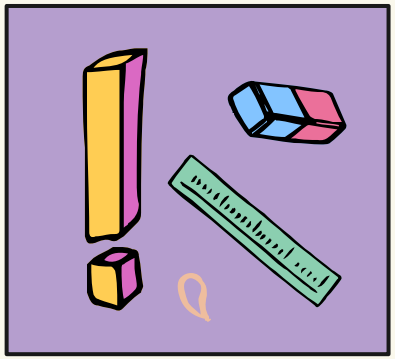
Minigame Subsystem

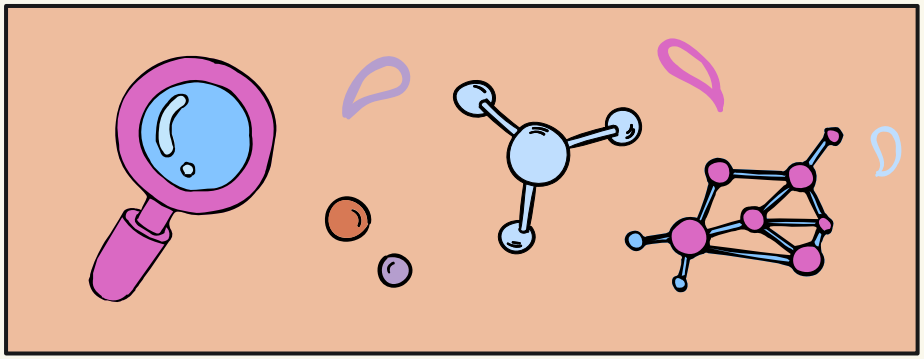
- Minigames developed independently for maximum **flexibility**
  - Set of requirements for **compatibility** with the main game
- Comprises interfaces such as:
  - **Minigame Access Interface**
    - Ensure minimum requirements are met for the user/minigame
    - Allows for methods to vary the relative difficulty of the levels
      - Works with **Minigame Data Interface** in Backend Subsystem
  - **Leaderboard Interface**
    - Loading and uploading minigame-specific scores

Candidate Architecture

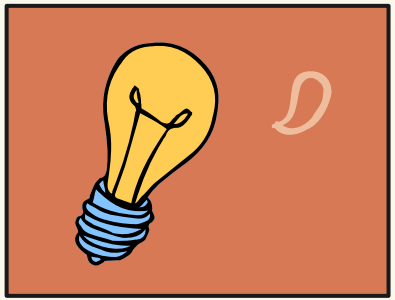
- **Layered Architecture**
- Clear flow and sequence of events
- Highly beneficial due to emphasis on **maintainability**
- Performance also a key consideration here
  - Single Point of Entry
  - Addresses potential race condition issues associated with having ‘Open’ layers







04

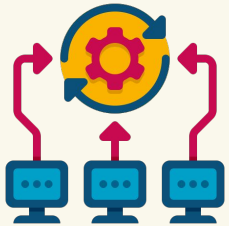


Testing

Testing Outline & Processes

Testing Approach

Unit
Testing

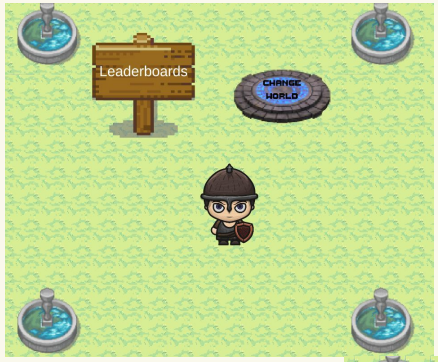


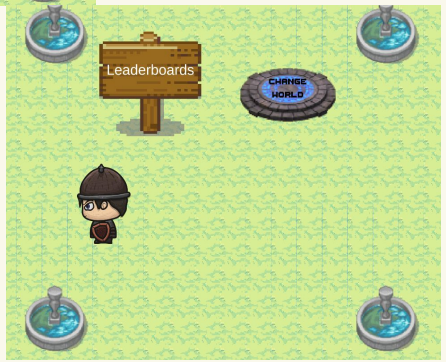
Integration
Testing

Systems
Testing

Unit Testing

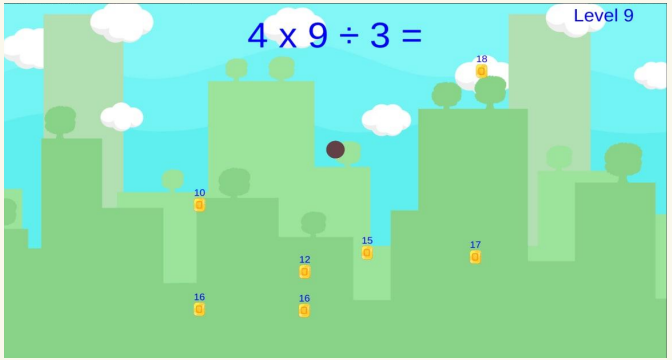
- Important for CastleSky as is highly dependent on the intended functionality of multiple components
  - Have to ensure these components work first
- Examples of unit testing include
  - User world navigation - Left movement
  - Student Menu navigation
  - Generation of access code





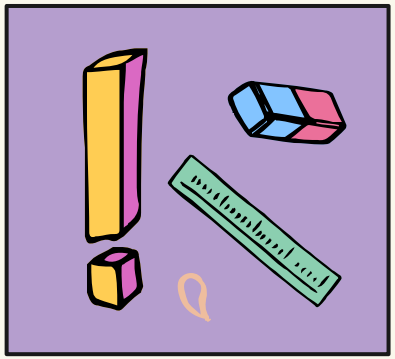
Integration Testing

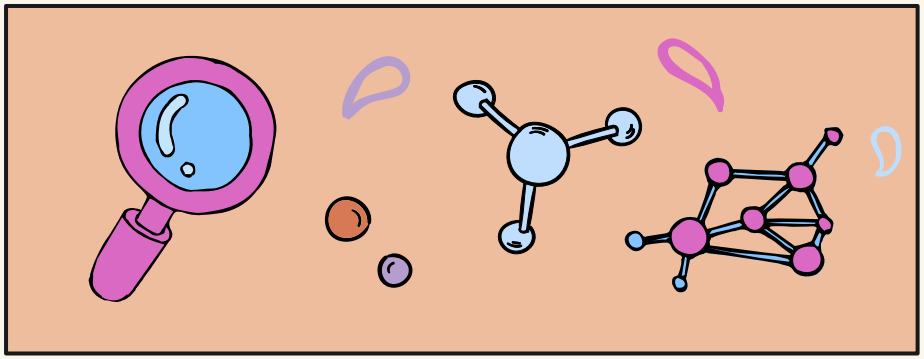
- Important considering the myriad of components
  - Components need to work together in tandem
- Minigames considered independent components, distinct from main game
  - Successful minigame integration a key part of development
- Examples of integrations testing include:
  - Minigame Entry
  - Retrieving Mini Game questions



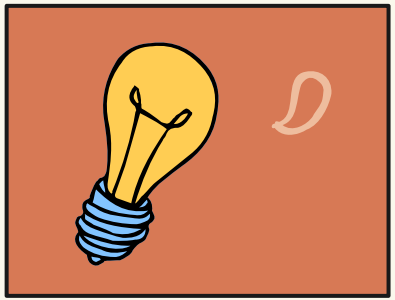
System Testing

- CastleSky is a game for children
  - System testing is a priority
  - Children tend to play games **a lot**
    - Have to ensure the game is capable of sustaining **rapid user inputs** and gameplay over sustained periods
    - Have to ensure the database is capable of handling **multiple requests and data updates**
- Examples of system testing include:
  - Smoke Test
  - Installation Test





04



Workload Distribution

Contributions to CastleSky

Development Team: Main Game

- Responsibilities
  - Main game user interface
  - Unit Testing for various components
  - Integration Testing for minigames
  - Backend integration with main game

Colin Tan

Min Ang

Development Team: Minigames

- Responsibilities
  - Minigame user interface
  - Unit Testing for various components
  - Minigame integration with main game
  - Backend integration with minigames

Calvin Low

Jeremy Teo

Justina Quak

Development Team: Backend

- Responsibilities
  - Set up and maintenance of backend server
  - Ensure maintainability and flexibility of API design structure
  - Ensure efficiency of database queries

Zhao Fengye

Justina Quak

Documentation Team

- Responsibilities
  - Delegate documentation components to team mates
  - Ensure development progress aligns with requirements and documentation
  - Ensure accuracy and relevance of diagrams

Manika Hennedige

Min Ang