CAMBRIDGE OCR

H446 A Level Computer Science

Creating concise NEA documentation

Connor Toms| 01 July 2026

Subject Advisor - Computer Science, IT and iMedia

Your Subject Advisors

Vinay Thawait

Computer Science, IT and iMedia

Connor Toms

Computer Science, IT and iMedia

Contact us: support@ocr.org.uk | 01223 553 998

The NEA
Overview

NEA Overview

- The NEA is an extended project that should **showcase** a candidates practical understanding of the programming content
- The NEA is worth 20% of the total assessment
    - This equates to approximately **72 hours** of directed learning
    - A similar ratio of out-of-classroom study
- The NEA uses a software development life cycle
    - Candidates may choose to use any software development model they wish
    - We would suggest one that encourages iterations

Project setting

- Candidates should choose a **significant problem** to solve
- However, we see many projects that scope problems which are well beyond A level standards
- Projects which are too large generally result in significant write-ups, which takes significant time to both create and mark
- Where candidates choose huge problems, it may be possible to focus on a **singular aspect** of this problem
- If in doubt we have our Project Setting Guidance service, which can review project proposals for suitability

CAMBRIDGE OCR

Project Setting Guidance

- Our **Project Setting Guidance** on Teach Cambridge **must** be read prior to asking students to create project ideas
- It covers topics and provides advice on areas like:
    - Project scope
    - Languages
    - Project types
    - Initial project ideas
- Candidates must have refined their ideas down to a single, concise and structured proposal for review by you (the teacher)
- **top tips for planning the NEA programming project blog**

CAMBRIDGE OCR

Proposal Reviews

- If you are still unsure about a proposal, then it may be sent to us to review
- Please check the following before sending them through!
    - You have not submitted similar projects in the past
    - You have reviewed all proposals yourself
    - You have only included proposals of concern
    - You have collated them all into a single document
- We receive hundreds of Centre's proposals – with many sending 20-30 proposals in at a time…

CAMBRIDGE OCR

Poor project proposals

- Many proposals which are sent to us are not suitable
- This is either down to:
  - Not having read the guidance beforehand or
  - Proposals are far too brief to be understood
- For example:

  Title: Road Heat

  Stakeholders: Friends and Family

  Language: Python

  Objectives: To create a 2D game where you dodge traffic running from the police.

  Why is this a suitable project: A niche gap in the 2D game market.

Poor project proposals

- This proposal is too brief
- It lacks detail about the game
- There are no mechanics involved
- It’s hard to visualise what the outcome is
- No mention of any algorithms

Proposal

Title: Road Heat

Stakeholders: Friends and Family

Language: Python

Objectives: To create a 2D game where you dodge traffic running from the police.

Why is this a suitable project: A niche gap in the 2D game market.

Poor project proposals

- This proposal is too brief
- It lacks detail about the game
- It presumes knowledge of other systems
- There are no mechanics involved
- No solid evidence to show how it will help hand-eye coordination
- No evidence this is a problem that ‘needs’ to be solved
- Links to ADHD and neurodiversity are tentative at best

Proposal

Title: Space Invaders for those with ADHD (Satellite Defence is the name I created)

Language: Python

Brief Overview: this project is a version of Space Invaders that tailors more to the needs of people who suffer from ADHD/ are neurodivergent

Main Objectives: To create a version of the game that can help those with ADHD/ are neurodivergent to increase hand-to-eye coordination, concentration

I think it's a suitable project as it is tackling a problem which many people struggle with and helping people with this problem can help them develop their skills to aid their learning and so that they can have the same fun with games as non-neurodivergent have

CAMBRIDGE OCR

Poor project proposals

- This proposal has a fair amount of detail
- It is very ICT oriented
- This is much more suited to SQL and a relational database
- Likely to lack and significant programming element
- Algorithms are in theory quite simple
- It attempts to mention automation but in a very limited way

Proposal

Title: Restaurant Inventory Management system

A software that helps a restaurant to manage their inventory efficiently based on stock control. The user will be able to enter the amount of stock and the system would then include a reorder level for the stock.

If the inventory reaches the reorder level, the user or the restaurant would get notifications. The user will also have the option to include the expiration date of the stock and the system will send notifications when the stock is nearing expiration.

The system aims to reduce wastage and manage stock efficiently.

Language: Python

Main project objectives/success criteria:

- Include a login system and main screen
- Provide real time and accurate data of inventory
- Provide graphs for each product in stock

Why I think this is a suitable: because it incorporates data analysis, data management and includes databases which allow data to be organised, accessed and manipulated in a structured way

Good project proposals

- Clear statement of problem
- Target audience
- Short overview of the mechanics
- Ideas on algorithms
- Explanation of logic chains
- Clear list of appropriate objectives
- Consideration of the programming that is done in Component 2 and how this could be used in the project

CAMBRIDGE OCR

Good project proposals

Title: Battle Chess

Audience: All ages

Aim: Create an AI to play a game of chess that has additional features to make it more creative. For example: pieces attack each other using combat mechanics, you can use power ups to heal pieces, add to attack power, increase armour etc. These are randomly generated to make the user choose between trying for a “standard” checkmate, or picking up power-ups to help them win

Algorithms: Random spawn generator for power ups, OOP for board and pieces, tree traversal algorithms for supporting AI. Prediction of board states. Analysis of how a power-up may affect the strength of a player

Objectives: Support and help tools, click and drag move of pieces, auto generation of power-ups, adaptive AI to create levels of difficulty.

Language: C# & Unity

Why is this suitable: Chess is a complex game to learn. It may feel less fun for younger players. I want to make this a more ‘arcade’ style game with more interesting feature to help younger players engage more with Chess and to give Chess a different feel for those who are more experienced.

Writing the NEA
Analysis Section

Analysis

- Project proposals can be developed into a short opening statement that clearly states the problem to be solved
- This **does not** need to be significant in length
- There are no specific mark criteria for identifying the problem
- Most of the detail will come later in the other sections of the analysis

Analysis

- This statement on the project is **concise**
- It frames the nature of the project clearly
- There is no additional information yet – as this comes **later**

Section 1 – Analysis

Identifying the Problem

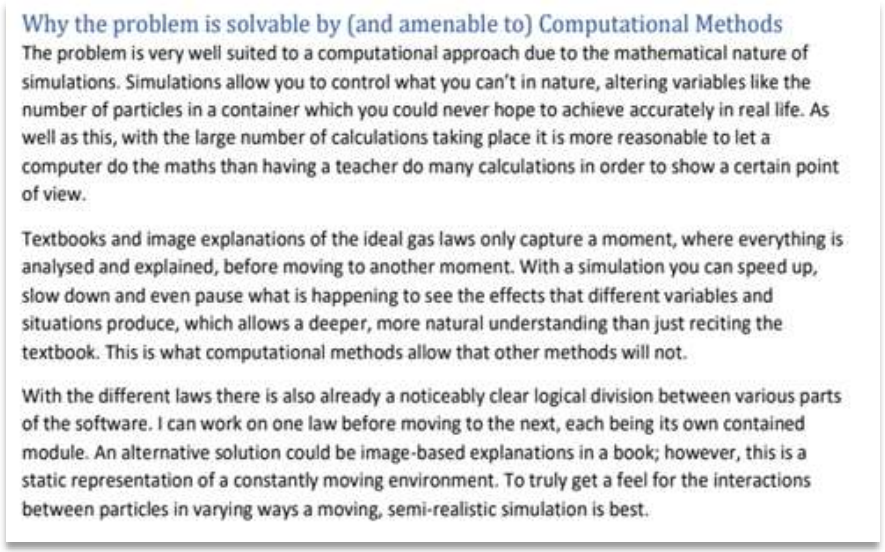
My programming project will be a simulation of the particles involved in the Ideal Gas Laws from the perspective of the physics A-Level OCR B syllabus. The ideal gas laws describe how temperature, pressure and volume interact in a container, along with the kinetic energy of the particles and their intermolecular forces. The definition of an ideal gas is one which obeys this equation of state at all pressures, volumes and temperatures.

$$pV = NkT$$

$$pressure * volume = No.of\ Particles * Boltzmann\ Constant * Temperature$$

Having done this topic recently I found many of my peers struggled with aspects of the Gas Laws such as thinking about why forces on the molecules change as they move towards the edge of the container. As well as this I have found in my research of the gas laws that the solutions already available do not go into depth and omit many tools that aid linking of the gas laws to what textbooks say on how they work.

Analysis



Why the problem is solvable by (and amenable to) Computational Methods

The problem is very well suited to a computational approach due to the mathematical nature of simulations. Simulations allow you to control what you can't in nature, altering variables like the number of particles in a container which you could never hope to achieve accurately in real life. As well as this, with the large number of calculations taking place it is more reasonable to let a computer do the maths than having a teacher do many calculations in order to show a certain point of view.

Textbooks and image explanations of the ideal gas laws only capture a moment, where everything is analysed and explained, before moving to another moment. With a simulation you can speed up, slow down and even pause what is happening to see the effects that different variables and situations produce, which allows a deeper, more natural understanding than just reciting the textbook. This is what computational methods allow that other methods will not.

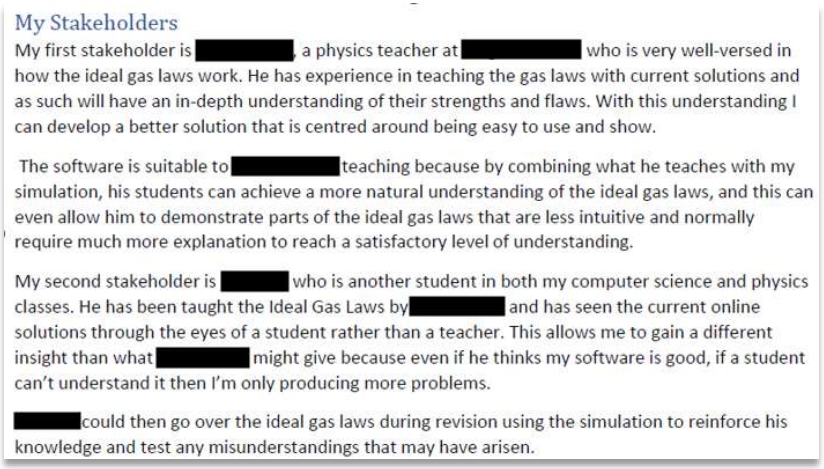
With the different laws there is also already a noticeably clear logical division between various parts of the software. I can work on one law before moving to the next, each being its own contained module. An alternative solution could be image-based explanations in a book; however, this is a static representation of a constantly moving environment. To truly get a feel for the interactions between particles in varying ways a moving, semi-realistic simulation is best.

Computational methods

- Candidates are **not** required to discuss what computational methods are
- They should describe how the project it **suitable** for a computational approach

Remember that…
Explaining computational methods adds significant length to a project and isn't required…

Analysis



My Stakeholders

My first stakeholder is [redacted], a physics teacher at [redacted] who is very well-versed in how the ideal gas laws work. He has experience in teaching the gas laws with current solutions and as such will have an in-depth understanding of their strengths and flaws. With this understanding I can develop a better solution that is centred around being easy to use and show.

The software is suitable to [redacted] teaching because by combining what he teaches with my simulation, his students can achieve a more natural understanding of the ideal gas laws, and this can even allow him to demonstrate parts of the ideal gas laws that are less intuitive and normally require much more explanation to reach a satisfactory level of understanding.

My second stakeholder is [redacted] who is another student in both my computer science and physics classes. He has been taught the Ideal Gas Laws by [redacted] and has seen the current online solutions through the eyes of a student rather than a teacher. This allows me to gain a different insight than what [redacted] might give because even if he thinks my software is good, if a student can't understand it then I'm only producing more problems.

[redacted] could then go over the ideal gas laws during revision using the simulation to reinforce his knowledge and test any misunderstandings that may have arisen.

Remember that…
Surveys can increase the length of a project significantly.
They produce good evidence for justifying features – but do add to workload. Use them sparingly.

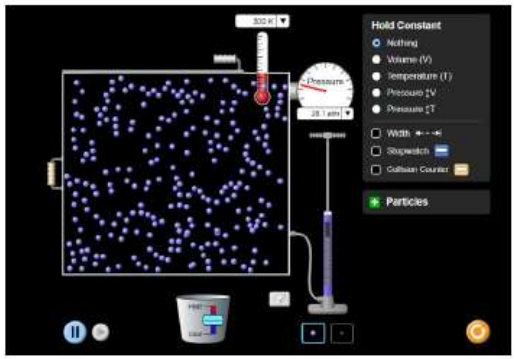
Stakeholders

- Candidates should have a **range** of stakeholders
- These should be clearly stated
- A short paragraph for each stakeholder which describes how they will **use** the system and how it is **appropriate** to their needs
- Stakeholder interviews may help define features – but **summarise** the outcomes
- Additional evidence may be added in an appendix

Analysis

My Research and Solutions to Similar Problems

There are 2 main Ideal Gas Laws at A-Level, Charles' Law and Boyle's Law, and visualisations of both have been made on websites such as Phet, however often they do not show the more in-depth workings of the gas laws and most importantly how they differ from real life.



The example above by a website called Phet has controls to change the temperature, a pump to add more particles, volume sliders and a pressure calculator. These features are all done well for what they are and, to quote Mr. Dawson, "It's good as an introduction," however it does not look at the assumptions of the ideal gases. As well as this the particles bouncing in the container are not actually used to calculate pressure and temperature, they are just a means of showing the effects of higher temperature for example. A real simulation would calculate pressure by measuring how many particles collide with the walls and the force with which those particles collide, then using force / area to calculate pressure rather than pressure = NkT/V. This then shows how real results differ from calculated ones.

The effects of intermolecular forces aren't explored here, showing how the particles in the centre get attracted to those around it, while the particles next to the walls have a net attraction away from the wall, affecting the pressure in the container.

This simulation has shown me that a better solution will at least talk about how the simulation relates to real-life, what the simulation is doing that real life isn't, and have more features to help students understand the gas laws as well. I will adjust my approach to the problem to be more focused on not just getting the particles colliding correctly, but explaining how the collisions happening may be different to that of real-life, but close enough to be a suitable model for students to learn from.

Research

- Candidates should investigate a **range** of other solutions or ideas
- We would expect a **summary** of 3 or more systems for top mark band
- If nothing similar exists, use systems that may use similar interfaces, or ideas, or themes etc.
- Tabulate **strengths** and **weaknesses** to keep the write-up **concise**
- Tabulate **key features** to take forwards.

Remember that...
Tables can help summarise findings quickly, clearly and concisely.

Analysis

The Essential Features

Feature	Description	Justification
Varying Heat, pressure, and volume variables	This would allow users to see the effects of changing temperature on the average kinetic energy of particles, pressure, and volume of a container. Users could also vary volume while keeping the same temperature, seeing what effect this might have on pressure.	Without changing the variables, it is hard to then understand the effects they have on each other. By being able to manipulate temperature and volume in ways not possible in real life, students can more easily see how the particles and variables interact.
Tracking one particle	Make every particle but one translucent / a different colour.	This allows users to look even further into the effects of changing variables. students can narrow their focus down to one particle and how that particle is affected, rather than the simulation as a whole.
Explanations of the gas laws	Next to the simulation is a small explanation of which gas law is being simulated and what is going on. These explanations can be switched between.	Allows students to relate the gas law on the screen to the one mentioned in textbooks, so that later they can think back to what was happening and have a more intuitive understanding.
Ability to add and remove particles	A quick and easy way of altering the number of particles.	Some situations are better taught with many particles, or only a few. This allows teachers to set up any situation which they believe will help the students to learn more.
Toggle between ideal / non-ideal state	Allows toggling of the simulation from what an ideal gas would look like to what a real gas would act like	Students can then see how real gases act vs ideal ones, seeing the effect this has on the calculations and forces present on each particle.
Calculations of Ideal Gas statistics	Calculations of Average Kinetic Energy, average speed and pressure.	These allow students to understand the more complex properties of the particles, and be able to form their own theories of why certain things happen in certain situations.

Remember that…
A clear link between research, discussion and outcomes makes the **evaluation** easier at the end of the project.

Key features

- Tables keep this section clear and concise
- Don't be afraid to use **bullets** to help reduce writing even further
- Justifications should **link** back to research and end user interactions for a real flow to the project
- Clear **justification** is clear for upper marks
- Consider splitting features into **essential** and **desirable** – especially for larger projects

Analysis

The Success Criteria

Success Criteria	Evidenced By	Justification
Have particles moving in the container	A screen recording of the particles moving around the container	Particles must move to have any effect on the container
Particles colliding with walls of the container	A screen recording of the particles bouncing off the container walls	Particles can then exert pressure on the container walls
Working collision detection between particles	Screen recording of particles bouncing off each other	Particles collide in both real life and in an ideal gas
Ability to add particles to the container	Screen recording showing particles being added	Nothing can happen in the container without particles being there.
Ability to remove particles from the container	Screen recording showing particles being removed.	Omission of this feature would force the user to restart the software each time they wanted to reset the container which is unnecessary.
Buttons to switch between the different ideal gas law explanations.	Screenshots of different descriptions being output	Students can then relate the description to what is being shown.
Ability to alter Volume	Screenshot of different container sizes	Effect of changing volume can be demonstrated to students
Ability to alter Temperature	Screen recording showing particles speeding up as temperature is increased	Effect of changing temperature can be shown.
Ability to switch between using the ideal gas assumptions and not using them	Screen recording of the behaviour of the particle changing when the toggle is enabled	Allows students to see where the textbook makes assumptions to simplify learning.
Force arrows showing intermolecular forces between particles (when tracking one particle)	Screenshot showing force arrows towards particles	This allows the simulation to go past what the other 2 solutions did, which Mr Lawson has stated would make the software a lot better.
Temperature, pressure or volume changing as a result of another variable being changed.	A screenshot of the calculations carried out by the computer and a worked example by Mr Lawson confirming the maths is correct (Stakeholder witness statement)	The calculations should be checked to ensure they are correct and not misleading. By showing the calculations students can see what is being used with what which will help their own understanding for exam questions.

Success criteria

- Must be **measurable**
- Tables help to **reduce** extended writing
- Be **specific** – as this will help the evaluation later
- Justifications must be **explicit** and should link back to user requirements where appropriate
- May be split into **core** and **additional** if this is a larger project

Remember that…

Many candidates don't make these measurable or justified. This means that the evaluation section can therefore suffer later…

Writing the NEA
Design Section

Concise designs

- A summary of the ideas on decompositions and how it has been used is fine
- But the approach **must** be both **explained** and **justified**

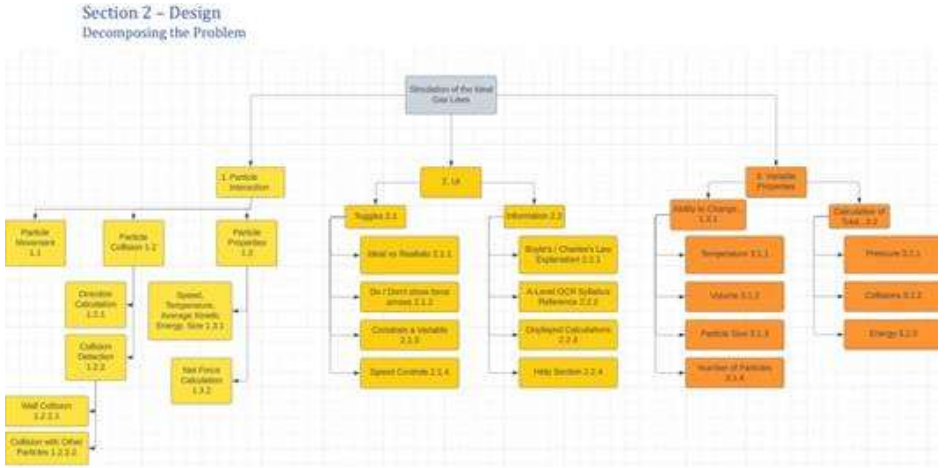
Why I've Broken My Program Down in This way.

To ensure that the software works as intended, I felt that I should break the simulation into three sections. The Particle interaction section is the most important for my software as if this doesn't work there is no point in having the rest, and decomposing this further allows me to work on each aspect of the particles (movement, collision detection and direction calculation) fully and to completion.

UI is separate to this and only involves sorting through the various explanations and statistics shown in the software, and for this reason I chose to make it a different section. Organisation is very important for this project as there are a lot of calculations being done that can be overwhelming if not sorted properly.

By breaking the simulation down in this way, I make the process more modular, which makes testing during development much easier, improves maintainability in the future, and also allows me to more easily organise scripts in the Unity project.

Particle Movement 1.1	The particles have energy which means they must be moving; 0 kelvin is not possible in ideal gases or real life. Speed will be a variable that all particles have, and will change depending on factors such as if the gas is ideal, what particles around it are doing, how close it is to the edge of the container (less intermolecular forces on container side of particle)
Wall collision 1.2.2.1	The particles must remain inside the container and so I need to detect when the particles touch the edges and then call the direction calculation function to calculate their new direction.
Collision with other particles 1.2.2.2	For ideal gases this will assume perfectly elastic collisions in which no energy is lost. In a real gas a small amount of energy will be lost in each collision and this small loss of energy to the surroundings will lead to the average kinetic energy decreasing over time.

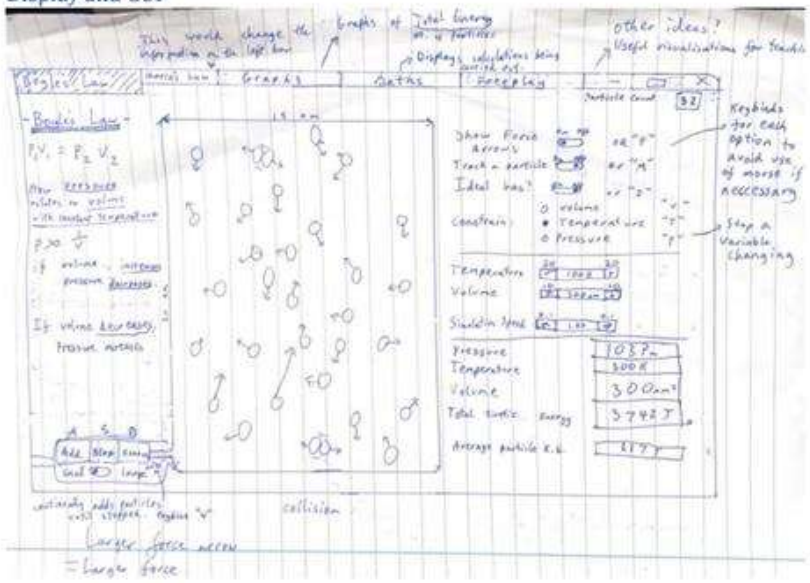


Remember that...
Using formal design tools can help create clear breakdowns which are easy to see and follow

Usability features

- Hand drawn designs for user interfaces are often **quicker**
- Annotations can be added to save writing in paragraphs
- The **key features** should be highlighted and commented on
- Key colour themes can be stated to cover all diagrams

Display and GUI



This is the main window of the program. I have added key-bindings to cater to people who may not have a mouse, and this can also allow faster, easier access to changing anything about the program. This is especially important as teachers often use laptops given by their school and accurate cursor movement on trackpads isn't always possible, which can lead to difficulty when trying to click smaller buttons such as the "constrain a variable" buttons.

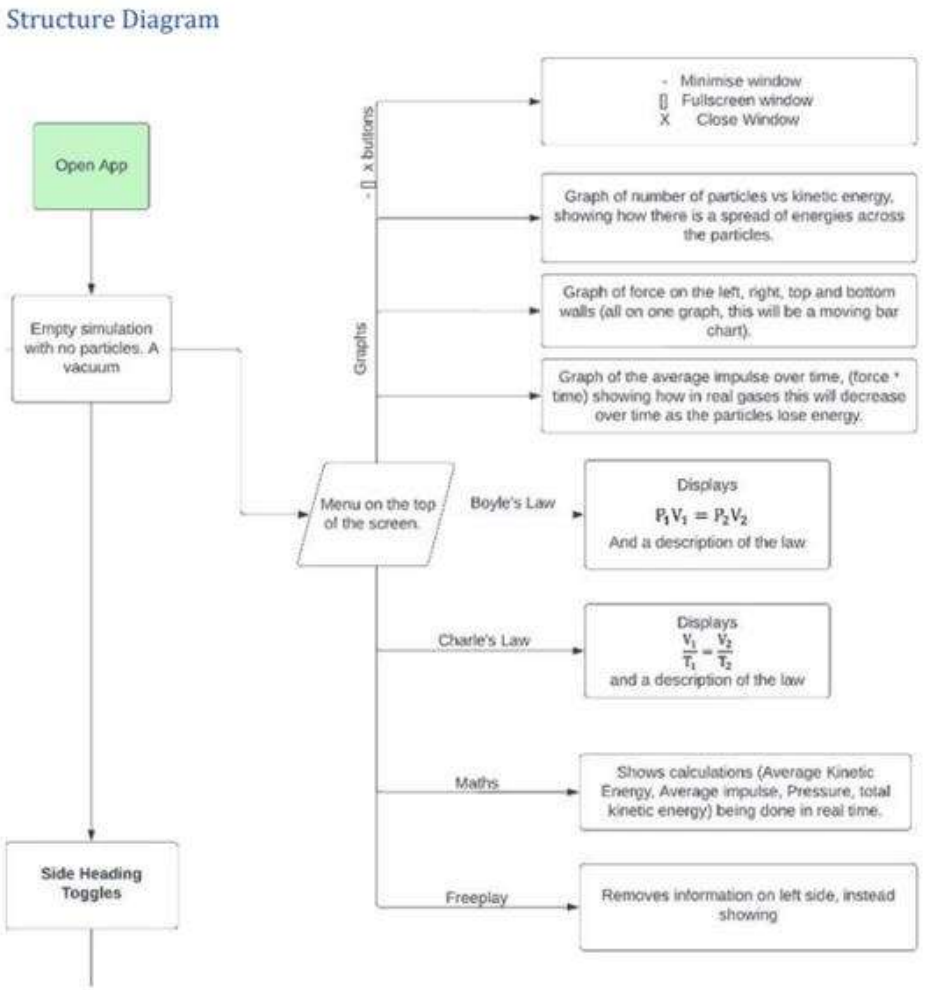
The buttons along the top will all enable a particular window on the left, disabling the currently active one, allowing different information to be shown. The toggles on the right show what can be changed or constrained in the simulation. The bottom right corner contains all the statistics about the simulation, tracking all or one of the particles with track-a-particle.

The top Headings

Boyle's Law – Boyle's Law equation and a quick explanation of what happens to pressure if volume increases / decreases with a constant temperature.

Structure design

- Again, **formal design** methods/tools often show information concisely
- Discussion around structure is fine, but using only prose will **extend** the length of a project



CAMBRIDGE OCR

Data Structures and variables

- All **key** variables and data structures should be listed
- It is often easier to do this for each **module/component** of the program rather than one huge table
- **Tabulating** data structures, variables and key information helps keep it simple
- Don't forget to include any **validation** which may be needed

Variable or Attribute	Name	Variable Type	Description
Attribute	Speed	float	Current speed of the particle
Attribute	Mass	int	Mass of the particle. This is one of 2 options, large or small, which will be 1.6735×10^{-24} grams (mass of hydrogen) or 2.6561×10^{-23} grams (mass of oxygen)
Variable	Temperature	int	Temperature of the container
Variable	Volume	int	Volume of the container. This is still in 2D, but volume makes more sense than area.
Variable	Kinetic Energy	float	Kinetic Energy of the particle
Variable	IsIdealGas	Boolean	Enables the ideal gas assumptions. Is true by default
Variable	trackAParticleIsOn	Boolean	Enables track a particle subroutine to run, described below.
Variable	showForceArrows	Boolean	Enables the force arrows subroutine to run, also described below.

Remember that...
It may be that some variables are added later as the program develops. This is fine to do. Note these in the Unit Recording Sheet to help guide the moderator.

Algorithms

- Stronger students may choose to use **pseudocode**
- Flowcharts are accepted alongside descriptions of an algorithms function
- Designs **should not** be reverse engineered
- These are designs and are likely to be **refined** once implementation starts
- It is unlikely that designs will survive 'first coding' attempts

Wall and Particle Collision Pseudocode

```
public collision(gameObject other, gameObject[] collidingParticles){
    if (other.tag == "particle"){
        reflectionAngle = direction.Dot(other.direction,
        this.gameObject.direction);
        direction.rotate(2*-reflectionAngle);
        // The dot product of two vectors gives the angle between
them, this angle can then be used to reverse the direction of the
particle.
        // reflect() isn't an actual method, it is a placeholder
for future math I will do during development.
    }
    else if(other.tag == "vertical"){
        other.gameObject.direction =
(other.gameObject.direction.x * -1, other.gameObject.direction.y)
        // VERTICAL WALL: multiplying by -1 reverses the
direction. y direction stays the same, x direction inverted. This
turns the particle around, same as a bounce.
    }
    else {
        other.gameObject.direction =
(other.gameObject.direction.x, other.gameObject.direction.y * -1)
        // HORIZONTAL WALL: multiplying by -1 reverses the
direction. x direction stays the same, y direction inverted. This
turns the particle around, same as a bounce.
    }
}
```

This function has 3 separate sections: vertical walls, horizontal walls, and particle collision. For horizontal and vertical walls, the bouncing is very easy, with a simple reverse of the x or y vector component of the particle.

For inter-particle collision of same-mass particles I take the vectors of the two particles and find the dot product of them. This gives me the angle of each particle to a plane between the two, which the particle's direction can then be rotated around.

Remember that...
Algorithm designs which are a copy and paste of final algorithms, and then edited slightly (reverse engineered) will **not** gain credit.

Test plans

- Test plans **must** cover both **qualitative** and **quantative** testing
- Use cases/scenario tests can be designed for end use acceptance
- It can be **helpful** to design tests on a module-by-module basis, based on decomposition/deigns
- Number the tests for easy reference later in the NEA documentation
- Test plans should include data validation and destructive tests

Remember that…
Test plans can be refined as and when designs change. This is also fine – and just needs referencing so that the moderator is aware of this when reviewing the NEA marking.

The Test Plan

Test Number	Description	Expected Outcome	Reasons for the test
1	Add different mass particles to container	Particles bounce and interact with each other, with the larger mass particles having more momentum than the smaller mass particles.	If the particles don't interact and bounce properly then everything else will be skewed as well.
2	In/de-crease temperature	Particles begin moving faster / slower; assuming increased temperature with volume constrained pressure increases, with pressure constrained volume increases.	This tests volume and pressure are calculated correctly and adjust the other accordingly to stay constant when needed.
3	In/de-crease volume	Container in/de-creases in size; assuming increased volume with pressure constrained temperature increases, with temperature constrained pressure increases	This tests pressure and temperature are calculated correctly and adjust the other accordingly to stay constant when needed.
4	Try Show Force Arrows	Arrows protrude from the particles (one if T.A.P. is on) showing the interacting forces and resultant force on the particle.	This is an important teaching point and one that is useful to understand properly. If it is done incorrectly then the simulation will be misleading.
5	Try Track a Particle (T.A.P)	All particles but one turn translucent.	Ensure it works as intended, and that the relevant statistics switch to recording that one particle.
6	Try toggling Ideal Gas	Particles start interacting differently. This requires a lot more calculations to ensure everything works considering friction, energy loss and random motion	This is one of the more complex parts of the software that requires lots of changes to the core particle interaction. For this reason, it is also where the most bugs are likely to occur, especially logic

Writing the NEA

Iterative Development and Testing to inform development

Building the iterative development log

- A **development diary** is a good way to evidence this section and to help keep it brief
- The diary should evidence **core parts** of the project's design and show:
    - Initial attempts
    - Key troubleshooting
    - Testing and refinement
    - The final outcome for that section
- Issues like minor syntax errors or minor logic errors **do not** need significant focus
- A screenshot of final, commented code with **annotations** can help reduce writing load

Remember that...
Copy and pasting from the original testing table helps ensure you don't miss tests. Make it clear where additional tests have been added for completeness.

Building the iterative development log

- A focus on creating the **iterations/sprints** as identified in the design helps to add continuity and clarity
- A **short** review of each stage should be carried out – summarising progress
- The development focal point of the project
- Cyclic nature of development is expected to be evidenced
- Write – test – refine
- Well commented code helps reduce writing prose to explain sections of the implemented solution and helps the moderator

Remember that…

Many candidates forget to log errors and test as they go. Encourage regular "stop and check" moments – every time they compile code, they are testing!

CAMBRIDGE OCR

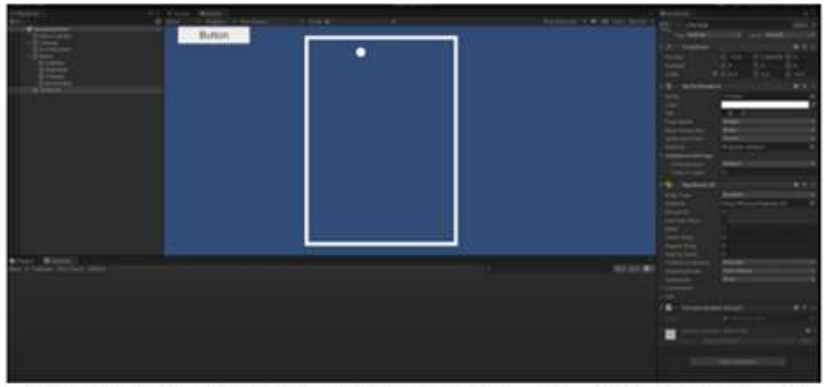
Building the iterative development log

- Here a candidate sets out the initial proposal for an iteration
- Early direction and success needs are stated
- An attempt to code it is then done

Section 3 - Development

Iteration 1 - Collision Detection and Optimisation

The fundamental objects are a circle sprite (a particle) and 4 rectangles acting as the walls of the container. The particles will have energy, move, collider with each other and the walls, and calculate pressure, average kinetic energy, average speed and more based on this. This will fulfil the functional requirement from my stakeholders of Colliding particles in a box.



The starting project. Nothing here works yet, but the particle has mass and walls it will bounce off later.

For the particles to move and be affected by other objects they need a Rigidbody2D component, which gives me access to the velocity, mass and gravitational scale of the particle. I chose to use a rigidbody2D instead of custom code as this handles a lot of the initial setup for me, leaving me more time to focus on the difficulties of the project such as direction and energy calculations. This will also allow the simulation to be more realistic than I could ever make with my own code.

Building the iterative development log

- The first attempt at the code is then completed
- Any other challenges or processes are logged
- A brief summary of progress is given
- Tests are then carried out on the code to check functionality

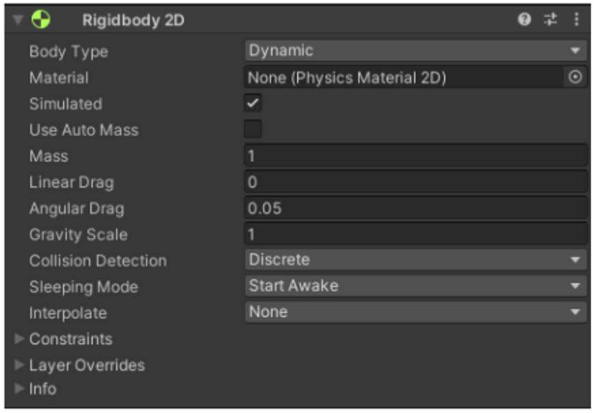
Remember that...
Adding additional code comments and annotation will help reduce the need for written prose.

It would be good here if the candidate had added some appropriate commenting to aid understanding and maintainability.

Wall Collision Detection

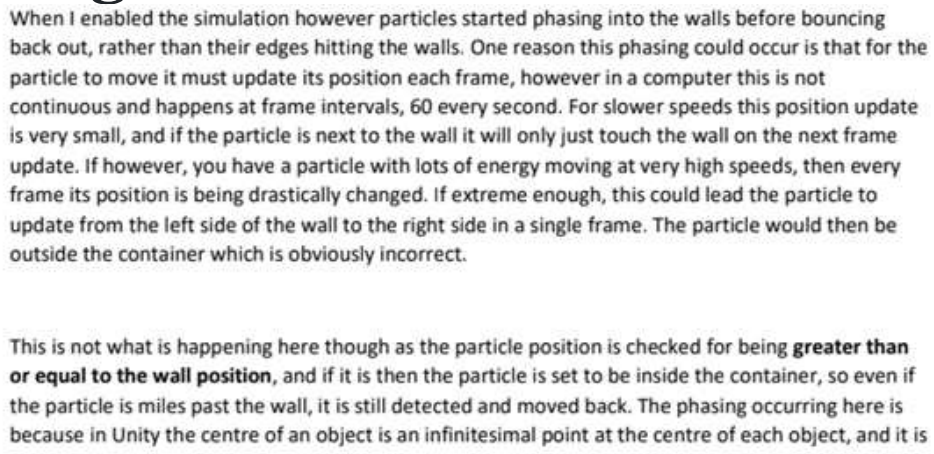
To let the particles calculate whether they are colliding with a wall or not I find the walls and their positions and then compare them to the particle's position by subtracting the particle x / y position from the wall's x / y position for the right and top walls, doing the inverse for the left and bottom walls. If the result is more (for right and top) / less (for left and bottom) than the particle's position, then I know a collision has happened and so print to the console "Collision detected", also setting the particle's position to the wall's position.

```
void checkWallCollisions()
{
    if (transform.position.y >= TopWall.transform.position.y)
    {
        transform.position = new Vector3(transform.position.x, TopWall.transform.position.y);
        rb.velocity = new Vector2(rb.velocity.x, rb.velocity.y * -1);
        Debug.Log("Wall Collision Detected");
    }

    else if (transform.position.y <= BottomWall.transform.position.y)
    {
        transform.position = new Vector3(transform.position.x, BottomWall.transform.position.y);
        rb.velocity = new Vector2(rb.velocity.x, rb.velocity.y * -1);
        Debug.Log("Wall Collision Detected");
    }

    else if (transform.position.x >= RightWall.transform.position.x)
    {
        transform.position = new Vector3(RightWall.transform.position.x , transform.position.y);
        rb.velocity = new Vector2(rb.velocity.x * -1, rb.velocity.y);
        Debug.Log("Wall Collision Detected");
    }
    else if (transform.position.x <= LeftWall.transform.position.x)
    {
        transform.position = new Vector3(LeftWall.transform.position.x, transform.position.y);
        rb.velocity = new Vector2(rb.velocity.x * -1, rb.velocity.y);
        Debug.Log("Wall Collision Detected");
    }
}
```



Building the iterative development log

- A **review** of the testing is then carried out
- It would **improve** this section if this review was:
  - Clearly linked to tests in the design section
  - Tabulated
  - Potential corrections are identified in table format rather than prose



When I enabled the simulation however particles started phasing into the walls before bouncing back out, rather than their edges hitting the walls. One reason this phasing could occur is that for the particle to move it must update its position each frame, however in a computer this is not continuous and happens at frame intervals, 60 every second. For slower speeds this position update is very small, and if the particle is next to the wall it will only just touch the wall on the next frame update. If however, you have a particle with lots of energy moving at very high speeds, then every frame its position is being drastically changed. If extreme enough, this could lead the particle to update from the left side of the wall to the right side in a single frame. The particle would then be outside the container which is obviously incorrect.

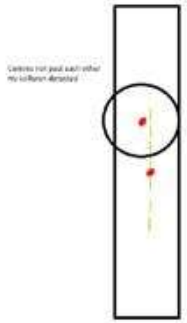
This is not what is happening here though as the particle position is checked for being **greater than or equal to the wall position**, and if it is then the particle is set to be inside the container, so even if the particle is miles past the wall, it is still detected and moved back. The phasing occurring here is because in Unity the centre of an object is an infinitesimal point at the centre of each object, and it is

Remember that…
We're looking for logs of significant challenges which affect the viability of the project – rather than small 'less significant' challenges like basic syntax errors

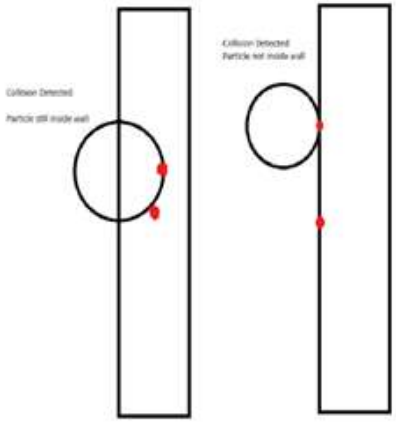
Building the iterative development log

- Here a more narrative correction is proposed using annotated diagrams

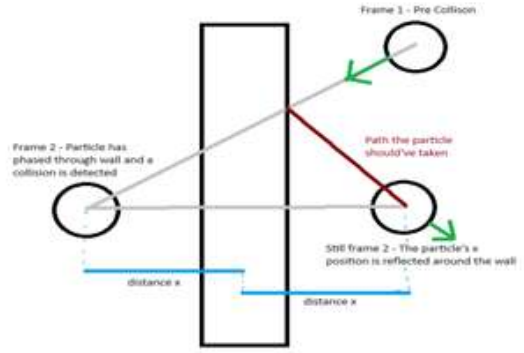
this point that is being compared to the particle's position. Even if the edge of the particle is inside the wall, the centres are not past each other and so nothing is detected.



To alleviate this, I added the radius of the particle to the check, which meant that this centre point was now shifted to the edge of the particle (below left image) and added the width of the wall (below right image), shifting the wall's centre point to the edge of the wall.



Once a collision is detected the particle position is updated taking the phasing into account as shown below.

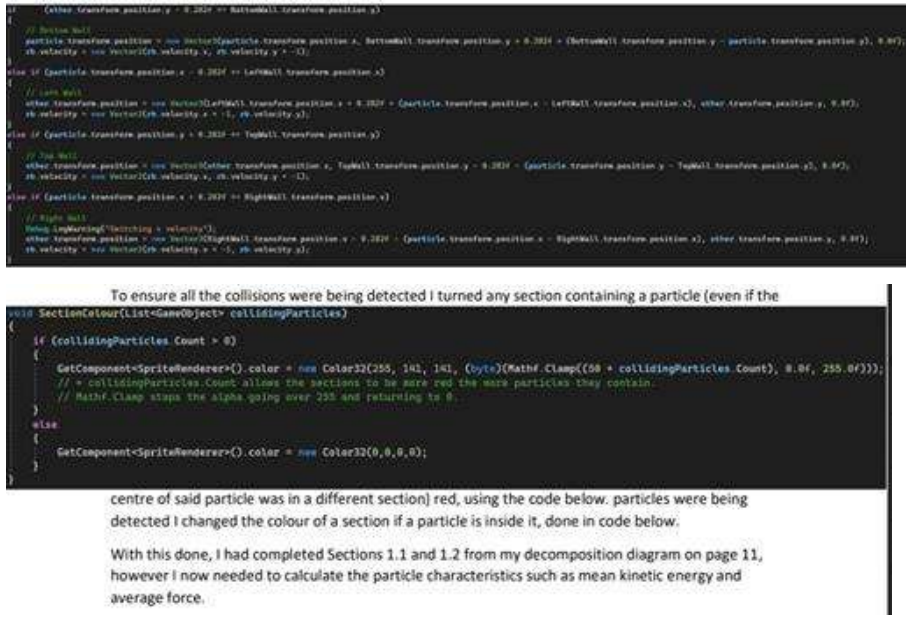


The code to take this phasing into account is shown below.

Remember that...
For this example – annotated diagrams work really well to show the problem and solution.

Building the iterative development log

- **Updated** code is then provided with additional commenting in-line within the code
- A short description of the changes are included
- Again – if appropriate – this could be presented in tabular format.



```
void SectionColour(List<GameObject> collidingParticles)
{
    if (collidingParticles.Count > 0)
    {
        GetComponent<SpriteRenderer>().color = new Color32(255, 141, 141, (byte)(Mathf.Clamp((50 + collidingParticles.Count), 0.0f, 255.0f)));
        // + collidingParticles.Count allows the sections to be more red the more particles they contain.
        // Mathf.Clamp stops the alpha going over 255 and returning to 0.
    }
    else
    {
        GetComponent<SpriteRenderer>().color = new Color32(0,0,0,0);
    }
}
```

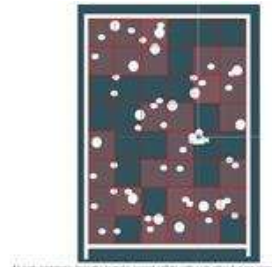
To ensure all the collisions were being detected I turned any section containing a particle (even if the centre of said particle was in a different section) red, using the code below. particles were being detected I changed the colour of a section if a particle is inside it, done in code below.

With this done, I had completed Sections 1.1 and 1.2 from my decomposition diagram on page 11, however I now needed to calculate the particle characteristics such as mean kinetic energy and average force.

Remember that…
This clearly shows improvements and direction of travel for the project.
It's clear that this now works – and the examiners are looking for evidence of a working solution.

Building the iterative development log

- An ‘iteration review’ is carried out to summarise the final success of that module/iteration.
- Unit testing is reviewed – using:
  - Initial tests from Design
  - Any additional tests that have been used as needed



An early prototype. Here the particles cannot collide with each other but are correctly bouncing off the walls. Each section is outlined in red however some particles aren't being detected.

To detect the particles, I used Unity's built-in method TriggerEnter2D, however this only works for 1 particle at a time. This meant some sections didn't turn red despite a particle being inside them, as shown in the above image. To fix this I instead used OverlapBoxAll which returns a list of all the colliders within that section, even if the centre of the particle is in a different section. This also allows for only particles near the walls of the simulation to be checked for wall collisions, reducing the number of calculations made.

This script worked better for multiple reasons.

- This script was added only to the coloured sections **on the edge** of the container, not the particles as was the case before. This means that the calculations are only carried out where necessary.
- This ran using OnTriggerStay2D instead of OnTriggerEnter2D, which runs the chosen code every frame for every collider that is touching it. This means as soon as the particle touches the respective section collider the script runs each frame checking for a wall collision.
- Finally, I increased the “TimeStep” in Unity’s settings from 0.01 to 0.02. This TimeStep alters how many times a script is run in a second, which I moved from every one frame to every two. This halves the number of times the script is run but does mean it will take longer to detect particles that are phasing through walls because we are checking less frequently. With the code taking phasing into account however this is not an issue.

Remember that…
It’s fine to add tests at this stage – especially if a refinement requires new variables or approaches to test its robustness.
These need to be highlighted clearly at this stage – and included in the URS to ensure the examiner can locate the additional evidence.

Building the iterative development log

- This process can then be repeated for the next iteration/development point
- Ensuring that tests are clearly linked back to the original designs are important
- This is easily done by simply copying parts of the testing tables into the iterative log and completing those tests
- Double check that all appropriate iterative/functionality test are completed by the end of the iterative development.

Remember that...
There may be some black box tests which take place after the system is completed to show the system working.
Some usability tests should be conducted with stakeholder input. This could be done during the iterative development if appropriate.

Writing the NEA

Testing to inform evaluation

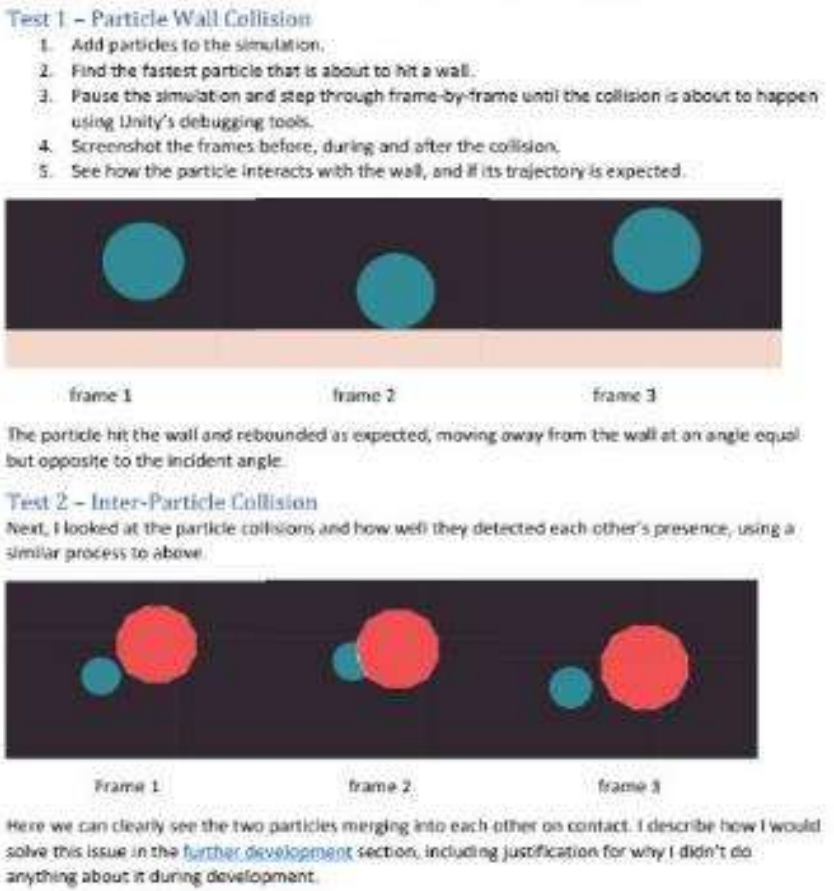
Testing to inform evaluation

- Testing in this section should include:
  - Whole system black box tests
  - Usability / stakeholder tests
- Using video evidence helps to show dynamics of a system, e.g.:
  - Menu transitions
  - Animations
  - Game play / functionality
- If using video evidence – keep it short and snappy!
- Use video name and time stamps in your testing table

Remember that…
There is no need for 4k video! Simple, short and 720p video should be more than enough to demonstrate tests.

Testing to inform evaluation

- Here a candidate has shown a black box test using screenshots
- This could also be done using a **video** – which would show the test taking place in a 'live' situation
- Stakeholder testing and white box testing can be done with '**use-cases**' or **scenario** testing



Test 1 – Particle Wall Collision

1. Add particles to the simulation.
2. Find the fastest particle that is about to hit a wall.
3. Pause the simulation and step through frame-by-frame until the collision is about to happen using Unity's debugging tools.
4. Screenshot the frames before, during and after the collision.
5. See how the particle interacts with the wall, and if its trajectory is expected.

frame 1 frame 2 frame 3

The particle hit the wall and rebounded as expected, moving away from the wall at an angle equal but opposite to the incident angle.

Test 2 – Inter-Particle Collision

Next, I looked at the particle collisions and how well they detected each other's presence, using a similar process to above.

Frame 1 frame 2 frame 3

Here we can clearly see the two particles merging into each other on contact. I describe how I would solve this issue in the further development section, including justification for why I didn't do anything about it during development.

CAMBRIDGE OCR

Testing to inform evaluation

- There is **no need** to replicate tests or copy and paste the entire testing table again
- Focus is only needed on those tests which need to still be completed
- Additional testing at the evaluation stage may be designed and implemented
- Remember to ensure this is **clearly added** to the URS to direct the moderator

Writing the NEA
Evaluation

Evaluation of success criteria

- Copy and paste of the original success criteria table with added **review** of whether it has been completely, partially or unsuccessfully met
- A **link** to evidence location is all that's needed
- Referencing test evidence is **important** and should be **explicit** – rather than (as we see here) slightly vague

Remember that…
There is no requirement for all success criteria to be fully met to reach the upper mark bands. However, the project should be mostly functional…

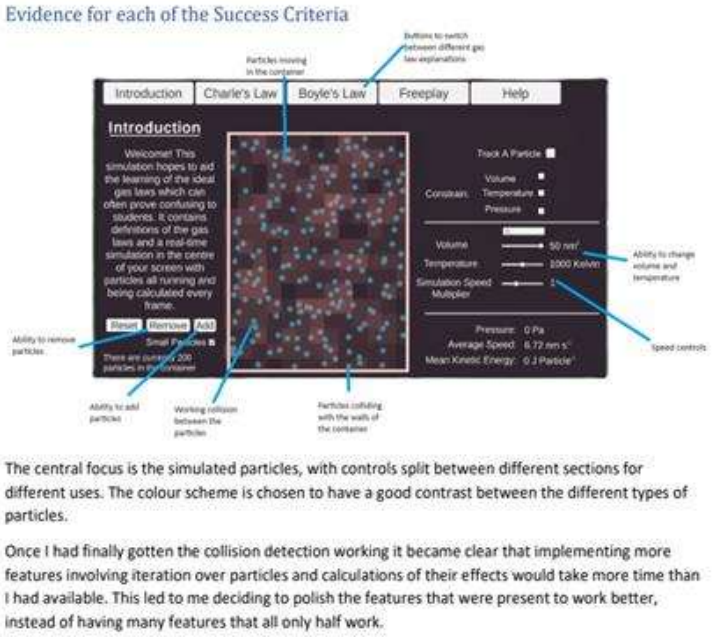
Section 5 - Evaluation

Evaluating How Many of the Success Criteria have been Met.

Success Criteria	Completed?	Evidenced Where?
Have particles moving in the container	Yes	Video linked below
Particles colliding with walls of the container	Yes	Video linked below
Working collision detection between particles	Yes	Video linked below
Ability to add particles to the container	Yes	Video linked below
Ability to remove particles from the container	Yes	Video linked below
Buttons to switch between the different gas law explanations.	Yes	Screenshot
Ability to alter Volume	Yes	Video linked below
Ability to alter Temperature	Yes	Video linked below
Ability to switch between using the ideal gas assumptions and not using them	No	-
Force arrows showing intermolecular forces between particles (when tracking one particle)	No	-
Temperature, pressure or volume changing as a result of another variable being changed.	Yes	Video linked Below
Track a Particle – All particles but 1 become translucent, with the pressure and particle statistics being based solely on that particle.	Yes	Screenshot

Evaluation of success criteria

- Annotated screenshots are also effective ways of presenting evidence
- These help to reduce prose and thus limit word count
- Usability features may also be evaluated in a tabular format listing:
    - Aim
    - Success
    - Impact



Remember that...
Candidates often go off on tangents in this section. Be clinical in evaluation the success of each criteria and avoid the waffle!

CAMBRIDGE OCR

Evaluation of success criteria

- **Limitations** of the system should be open and honest
- Avoid discussion of ‘lack of time’ or ‘lack of skills’ as a limitation
- Candidates must write with an **evaluative** approach. Rather than just describing the limitations
- Maintenance limitations/developments are project specific.
- Ideas such as the following could be included if appropriate:
    - Code maintenance
    - Ease of developing in the future
    - Server maintenance
    - Updates
    - Backups

Summary points

Key points

- Select your projects with some level of **measure**
- Huge systems = more documentation
- Avoid use of **repeat skills** (e.g. multiple similar levels in a game). This creates more documentation around the same skills/ideas.
- Keep things **concise** by using:
    - Bulleted lists
    - Design tools
    - Tabulation
    - Video evidence
    - Avoid repetition
- Use **mark scheme headings** to ensure evidence is clearly referenced and thus also avoids repetition
- Use the URS to **clearly point** moderator to **core evidence** for each marking bullet

CAMBRIDGE OCR

Thank You

