CCST9064 Group L – Video Essay
Individual Research Log

Student Information

Name:	SHING, Zhan Ho Jacob	**Group Number:**	L
Student ID:	3036228892	**Topic:**	Somatic Gene Therapy

1 Research Approach and Methodology

For this group project, I am not responsible for the research part. However, for video production purposes, research was still an essential part to obtain contents and ensure my content's accuracy.

The sources that I have consulted, as will be listed in the later sections, are mainly from credible scientific database – the Protein Data Bank (PDB) – and educational videos from reputable channels on YouTube. The research methodology mainly involves keyword searching within PDB and YouTube. The keywords are taken from the video script written by the scriptwriters and from the storyboard.

2 Key Sources Consulted

As our video essay is about highly specialised scientific topic – somatic gene therapy – there are many credible sources that can be used to explain the concepts clearly visually. Using existing materials also saves the production time and reduces the need for creating complicated animations *de novo*. I have researched and collected the following materials:

2.1 3D Models of Protein Structures

Many of the protein structures are already determined through experimental methods and are available in the Protein Data Bank (PDB, https://www.rcsb.org/). Further, for dynamically showing the structures, the molecular data downloaded from PDB can be fed into the software ChimeraX (https://www.cgl.ucsf.edu/chimerax/), colour-coded, and exported as a 3D object file, that can be further animated with After Effects.

In our video essay, the following protein structures were (adapted and) used:

- [1APL] *Crystal Structure of a MAT-α2 Homeodomain-Operator Complex Suggests a General Model for Homeodomain-DNA Interactions* – the double helix DNA structure was extracted for the opening scene.
- [2HHB] *The Crystal Structure of Human Deoxyhaemoglobin at 1.74 Angstroms Resolution* – the normal human haemoglobin structure, used along with 2HBS.
- [2HBS] *The High Resolution Crystal Structure of Deoxyhemoglobin S* – the sickle-cell haemoglobin structure, used along with 2HHB to illustrate the molecular basis of sickle-cell anaemia.
- [3FSN] *Crystal Structure of RPE65 at 2.14 Angstrom Resolution* – to illustrate the protein involved in gene therapy for Leber's Congenital Amaurosis (LCA).
- [4QQ6] *Crystal Structure of tudor domain of SMN1 in complex with a small organic molecule* – to illustrate the protein involved in gene therapy for Spinal Muscular Atrophy (SMA).

Apart from the models provided by PDB, there was also one double helix DNA model created by the software ChimeraX, using a randomly generated DNA sequence, for animating the rotating double helix DNA in the video.

2.2 Video/Animation Clips

To illustrate the complex biological processes involved in somatic gene therapy, I have searched for existing video/animation clips that can be used directly or adapted for our video essay. The following clips were used:

- *3D Animation of a Clogged Blood Vessel Due to a Sickle Cell Disease* (https://youtu.be/BmnfR-D8ewE) – to illustrate the effect of sickle-cell anaemia blocking blood vessels.
- *Gene Therapy Basics (2022 Update)* (https://www.youtube.com/watch?v=kAtd9X29SdQ) – to illustrate "delivering a healthy gene into a cell as compensation".

- *Breast Cancer* (https://vimeo.com/871843858) – to illustrate the idea of uncontrolled cell growth in cancer.
- *Introduction to CRISPR-Cas9 Genome Editing* (https://www.youtube.com/watch?v=iEA-NleJoqY) – to illustrate the CRISPR-Cas9 gene editing mechanism.
- *A Look at How CAR-T Cell Therapy Works* (https://www.youtube.com/watch?v=mXADrg_ckhI) – to illustrate the CAR-T cell therapy mechanism.

3 Major Insights and Discovery

As opposed to my original expectation that all of the protein structures in our body can be found on PDB as long as their genetic sequences are known, during the research process, I have discovered this to be false. Many of the protein structures cannot be found on PDB. Later investigations on this issue revealed that the structure of a protein cannot be determined simply through mathematical or computer simulations even if the exact amino acid chain is known. This is because protein folding is a complex process and is affected by many factors.

The truth is, many of the existing protein structures in PDB, except for those simple peptides, are determined through experimental methods. This can be verified because many of the PDB entries are actually a protein in complex with other molecules, such as ligands, inhibitors, or is a protein dissolved in a solvent, not the pure protein alone.

4 Personal Contribution to the Project

My responsibilities in this group project is to realise the storyboard designed by Jialiu Xu by assembling the video clips, animations, voiceovers together into the final video.

4.1 My Workflow for Contributing to the Video Essay

After receiving the storyboard from Jialiu Xu, I started creating the animation clips that were implementable using Python with the Manim library. The more complex animations, such as the sickle cell anaemia animations, were looked up online and directly used in our video with proper in-video citation. Those videos were not created because the complexity is beyond the requirement of this course.

With Manim, a total of 26 scenes were created (of which only 25 were used as one of them was no longer applicable after some modifications). Some of the designs in the original storyboard were not adopted due to technical limitations and time constraints.

As for the protein structure models, I fetched them from PDB into ChimeraX. With the built-in functions of ChimeraX, I colour-coded the structures (mainly by chain) and exported them as `.glb` 3D object files. These files were then imported into Adobe After Effects for simple animating (mainly rotation and zooming).

The animation clips, B-roll footages, 3D animations, and voiceover recordings were assembled together in Adobe Premiere Pro. Some simple video effects, such as zooming in/out, panning, and fading transitions were also applied in Premiere Pro to enhance the visual experience.

4.2 Animations Created with Python

Various animation clips were created using Python with the library Manim. The source codes are attached at the end of this document. For the sake of academic integrity, it is declared that large language models were involved in writing the following code files. The use of LLMs was mainly for handling complex mathematical expressions to save up time for actually producing the animations.

4.3 Leadership Role in the Group

Initially, two members were assigned the role as video production. This is impractical as the video editing software cannot be used collaboratively and only one person can work on the project files at a time. Uploading the project files to a cloud storage for sharing was also not an option as the files can get very large (several gigabytes) and the upload/download time would be too long.

Therefore, for easier task delegation, I took the initiative to offer the other video production member to focus on creating the storyboard from the script. This is effective since I, as a Computer Science student, find

it difficult to come up with creative visual designs from scratch on top of the script. However, with my previous experience in video editing, I can effectively realise the storyboard into the final video. Therefore, we agreed on this division of labour, which greatly improved our productivity and was a successful collaboration.

5 Reflection on Learning

5.1 Understanding of Genetics

As I was previously enrolled as an MBBS student, I already have some foundational knowledge of genetics through trainings from the IASM block. This has equipped me with the necessary background and skills to understand the complex concepts involved in somatic gene therapy. On the science side, through this project, I have additional knowledge about the specific applications of somatic gene therapy, such as Luxturna for LCA, Nusinersen for SMA, etc. On the moral side, I have also learned about more ethical considerations from more aspects, such as public opinion and fair access to treatment.

5.2 Skills Acquired

By working on this project, I have deepened my skills in professional video production. I have also learned new skills in 3D molecular visualisation and animation using ChimeraX and Adobe After Effects. Further, in 2D animation creation, I have improved my skills in using Python with the Manim library to create mathematical and scientific animations programmatically.

Annex: Python Source Codes for Creating Animations

`intro_scenes.py`

This file contains code for creating animations used in the introduction section of the video.

```python
from manim import *

# INTRODUCTION SCENES

class Scene1_TitleText(Scene):
    """Opening shot: Somatic Gene Therapy"""
    def construct(self):
        # Text animation
        text = Tex("Science Fiction", font_size=48, color=GREEN)
        self.play(Write(text))
        self.wait(2)

        newText = Tex("Somatic Gene Therapy", font_size=48, color=RED)
        self.play(Transform(text, newText))
        self.wait(3)

        self.play(FadeOut(text))

class Scene2_FixingGeneErrorsInACell(Scene):
    """Scene: Fixing Gene Errors in a Cell"""
    def construct(self):
        # Create cell (outer circle) with background
        cell = Circle(radius=2.5, color=BLUE, stroke_width=3, fill_color=BLUE, fill_opacity=0.3)

        # Create nucleus (inner circle) - smaller size with background
        nucleus = Circle(radius=0.9, color=PURPLE, stroke_width=3, fill_color=PURPLE,
        ↪  fill_opacity=0.35)

        # Create problematic DNA (red double helix)
        # Simple representation using two intertwined curves
        dna_strand1 = ParametricFunction(
            lambda t: np.array([
```

```
            0.3 * np.cos(2 * PI * t),
            0.8 * t - 0.4,
            0
            ]),
            t_range=[0, 1],
            color=RED
        )
        dna_strand2 = ParametricFunction(
            lambda t: np.array([
            -0.3 * np.cos(2 * PI * t),
            0.8 * t - 0.4,
            0
            ]),
            t_range=[0, 1],
            color=RED
        )
        problematic_dna = VGroup(dna_strand1, dna_strand2)

        # Create text labels
        cell_label = Tex("$\\textbf{Cell}$", font_size=28, color=BLUE).next_to(cell, DOWN, buff=0.3)
        problematic_dna_label = Tex("\\textbf{Problematic \\\\DNA}", font_size=24,
        ↪  color=RED).next_to(problematic_dna, RIGHT, buff=0.7)

        # Display cell, nucleus, problematic DNA, and labels at the start
        self.add(cell, nucleus, problematic_dna, cell_label, problematic_dna_label)
        self.wait(2)

        # Create fixed DNA (green double helix)
        fixed_strand1 = ParametricFunction(
            lambda t: np.array([
            0.3 * np.cos(2 * PI * t),
            0.8 * t - 0.4,
            0
            ]),
            t_range=[0, 1],
            color=GREEN
        )
        fixed_strand2 = ParametricFunction(
            lambda t: np.array([
            -0.3 * np.cos(2 * PI * t),
            0.8 * t - 0.4,
            0
            ]),
            t_range=[0, 1],
            color=GREEN
        )
        fixed_dna = VGroup(fixed_strand1, fixed_strand2)

        # Create fixed DNA label
        fixed_dna_label = Tex("\\textbf{Fixed \\\\DNA}", font_size=24, color=GREEN).next_to(fixed_dna,
        ↪  RIGHT, buff=1.0)

        # Flash effect
        flash = Flash(problematic_dna.get_center(), color=YELLOW, flash_radius=0.8)

        # Morph DNA to green with pop effect and morph the label
        self.play(
            Transform(problematic_dna, fixed_dna),
            Transform(problematic_dna_label, fixed_dna_label),
            flash,
            problematic_dna.animate.scale(1.3).set_color(GREEN)
        )
        self.play(problematic_dna.animate.scale(1/1.3))
        self.wait(2)
```

```python

class Scene3_NoPassing(Scene):
    """Scene: No Passing"""
    def construct(self):
        # Start with the ending status of Scene 2
        cell = Circle(radius=2.5, color=BLUE, stroke_width=3, fill_color=BLUE, fill_opacity=0.3)
        nucleus = Circle(radius=0.9, color=PURPLE, stroke_width=3, fill_color=PURPLE,
        ↪  fill_opacity=0.35)

        # Fixed DNA (green double helix)
        dna_strand1 = ParametricFunction(
            lambda t: np.array([
            0.3 * np.cos(2 * PI * t),
            0.8 * t - 0.4,
            0
            ]),
            t_range=[0, 1],
            color=GREEN
        )
        dna_strand2 = ParametricFunction(
            lambda t: np.array([
            -0.3 * np.cos(2 * PI * t),
            0.8 * t - 0.4,
            0
            ]),
            t_range=[0, 1],
            color=GREEN
        )
        fixed_dna = VGroup(dna_strand1, dna_strand2)

        # Create labels
        cell_label = Tex("$\\textbf{Cell}$", font_size=28, color=BLUE).next_to(cell, DOWN, buff=0.3)
        fixed_dna_label = Tex("\\textbf{Fixed \\\\DNA}", font_size=24, color=GREEN).next_to(fixed_dna,
        ↪  RIGHT, buff=1.0)

        # Group the cell components
        cell_group = VGroup(cell, nucleus, fixed_dna)

        # Add everything at the start
        self.add(cell_group, cell_label, fixed_dna_label)

        # Fade out the text labels
        self.play(FadeOut(cell_label), FadeOut(fixed_dna_label), run_time=0.25)

        # Create stick figures
        # Parent (taller, on the left)
        parent_head = Circle(radius=0.3, color=WHITE, stroke_width=2)
        parent_body = Line(start=ORIGIN, end=DOWN * 1.5, color=WHITE, stroke_width=2)
        parent_arms = Line(start=LEFT * 0.5, end=RIGHT * 0.5, color=WHITE, stroke_width=2)
        parent_legs = VGroup(
            Line(start=ORIGIN, end=DOWN * 0.8 + LEFT * 0.4, color=WHITE, stroke_width=2),
            Line(start=ORIGIN, end=DOWN * 0.8 + RIGHT * 0.4, color=WHITE, stroke_width=2)
        )

        parent = VGroup(parent_head, parent_body, parent_arms, parent_legs)
        parent_head.next_to(parent_body.get_start(), UP, buff=0.1)
        parent_arms.next_to(parent_body.get_start(), DOWN, buff=0.3)
        parent_legs.next_to(parent_body.get_end(), DOWN, buff=0)

        parent.move_to(LEFT * 3 + DOWN * 0.5)
        parent_label = Tex("\\textbf{Parent}", font_size=24, color=WHITE).next_to(parent, DOWN,
        ↪  buff=0.3)

        # Child (shorter, on the right)
```

```
        child_head = Circle(radius=0.25, color=WHITE, stroke_width=2)
        child_body = Line(start=ORIGIN, end=DOWN * 1.0, color=WHITE, stroke_width=2)
        child_arms = Line(start=LEFT * 0.4, end=RIGHT * 0.4, color=WHITE, stroke_width=2)
        child_legs = VGroup(
            Line(start=ORIGIN, end=DOWN * 0.6 + LEFT * 0.3, color=WHITE, stroke_width=2),
            Line(start=ORIGIN, end=DOWN * 0.6 + RIGHT * 0.3, color=WHITE, stroke_width=2)
        )

        child = VGroup(child_head, child_body, child_arms, child_legs)
        child_head.next_to(child_body.get_start(), UP, buff=0.1)
        child_arms.next_to(child_body.get_start(), DOWN, buff=0.2)
        child_legs.next_to(child_body.get_end(), DOWN, buff=0)

        child.move_to(RIGHT * 3 + DOWN * 0.3)
        child_label = Tex("\\textbf{Child}", font_size=24, color=WHITE).next_to(child, DOWN, buff=0.3)

        # Fade in stick figures and their labels
        self.play(
            FadeIn(parent), FadeIn(parent_label),
            FadeIn(child), FadeIn(child_label),
            cell_group.animate.scale(0.3).shift(parent.get_center() + RIGHT * 1.2),
            run_time=1
        )

        # Move cell next to parent
        self.play(cell_group.animate.move_to(parent.get_center() + RIGHT * 1.2), run_time=0.5)
        self.wait(0.25)

        # Draw arrow from left to right
        arrow = Arrow(
            start=LEFT * 1.5,
            end=RIGHT * 1.5,
            color=YELLOW,
            buff=0,
            stroke_width=6
        ).move_to(DOWN * 0.5)
        self.play(Create(arrow), run_time=0.25)

        # Cell attempts to move along arrow but is stopped by red cross
        # Create red cross in the middle
        cross_line1 = Line(start=UP * 0.3 + LEFT * 0.3, end=DOWN * 0.3 + RIGHT * 0.3, color=RED,
        ↪  stroke_width=8)
        cross_line2 = Line(start=UP * 0.3 + RIGHT * 0.3, end=DOWN * 0.3 + LEFT * 0.3, color=RED,
        ↪  stroke_width=8)
        red_cross = VGroup(cross_line1, cross_line2).move_to(DOWN * 0.5)

        # Animate cell moving and being stopped
        self.play(
            cell_group.animate.move_to(cell_group.get_center() + RIGHT * 1.5),
            run_time=1,
            rate_func=rush_into
        )

        # Show red cross appearing and cell bouncing back slightly
        self.play(Create(red_cross), cell_group.animate.shift(LEFT * 0.3), run_time=0.5)
        self.wait(2)
```

---

application_eyes.py

This file contains code for creating animations about gene therapy applications for eye diseases, specifically Luxturna treatment for Leber's Congenital Amaurosis (LCA).

```python
from manim import *

# APPLICATION - EYES SCENES

class Scene4_Chapter1Title(Scene):
    """Chapter Title: Application - Eyes"""
    def construct(self):
        # Text animation
        chapter = Tex("{\\scshape\\bfseries CHAPTER I}", font_size=50, color=YELLOW)
        title = Tex("{\\bfseries Genetic Blindness}", font_size=64, color=YELLOW)
        separator = Line(start=LEFT * 3, end=RIGHT * 3, color=ORANGE, stroke_width=4)

        chapter.move_to(UP * 0.75)
        title.move_to(DOWN * 0.75)
        separator.move_to(ORIGIN)

        self.play(
            Write(chapter, run_time=1.5),
            Create(separator, run_time=1.5),
            Write(title, run_time=1.5)
        )

        self.wait(3)

class Scene5_LuxturnaAndLCATitle(Scene):
    """Title: Luxturna and LCA"""
    def construct(self):
        # Text animation
        title = Tex("{\\bfseries Luxturna}", font_size=64, color=BLUE)
        subtitle = Tex("A gene therapy for", font_size=36)
        disease = Tex("{\\bfseries Leber Congenital Amaurosis (LCA)}", font_size=40, color=RED)

        title.move_to(UP * 1)
        subtitle.move_to(DOWN * 0.2)
        disease.move_to(DOWN * 0.8)

        self.play(Write(title, run_time=0.75))
        self.wait(3)
        self.play(Write(subtitle), Write(disease), run_time=0.75)

        self.wait(3)

class Scene6_LuxturnaPacking(Scene):
    """Luxturna Packing Animation"""
    def construct(self):
        scene_title = Tex("{\\bfseries Luxturna Mechanism}", font_size=48, color=YELLOW)
        scene_title.move_to(UP * 3)
        self.play(Write(scene_title), run_time=1)
        self.wait(0.5)

        # 1. Create cDNA (green single strand) on the left
        cdna = ParametricFunction(
            lambda t: np.array([
                0.2 * np.sin(4 * PI * t),
                2 * t - 1,
                0
            ]),
            t_range=[0, 1],
            color=GREEN,
            stroke_width=6
        )

```

```
        cdna_label1 = Tex("\\textbf{cDNA}", font_size=28, color=GREEN)
        cdna_label2 = Tex("\\textbf{(functional RPE65)}", font_size=24, color=GREEN)
        cdna_labels = VGroup(cdna_label1, cdna_label2).arrange(DOWN, buff=0.2)

        cdna.move_to(LEFT * 4)
        cdna_labels.next_to(cdna, DOWN, buff=0.5)

        # 2. Create AAV viral vector (purple capsid) on the right
        # Represent as a hexagonal/circular capsid with inner space
        aav_capsid = RegularPolygon(n=6, radius=1.2, color=PURPLE, stroke_width=4, fill_color=PURPLE,
        ↪  fill_opacity=0.2)

        # Add spike proteins around the capsid
        spike_positions = [
            UP * 1.2,
            UP * 0.6 + RIGHT * 1.0,
            DOWN * 0.6 + RIGHT * 1.0,
            DOWN * 1.2,
            DOWN * 0.6 + LEFT * 1.0,
            UP * 0.6 + LEFT * 1.0
        ]

        spikes = VGroup()
        for pos in spike_positions:
            spike = Triangle(color=PURPLE, fill_color=PURPLE, fill_opacity=0.8)
            spike.scale(0.15)
            spike.move_to(pos)
            # Point spike outward from center
            angle = np.arctan2(pos[1], pos[0])
            spike.rotate(angle + PI/2)
            spikes.add(spike)

        aav_vector = VGroup(aav_capsid, spikes)

        aav_label = Tex("\\textbf{AAV therapeutic\\\\vector}", font_size=28, color=PURPLE)

        aav_vector.move_to(RIGHT * 4)
        aav_label.next_to(aav_vector, DOWN, buff=0.5)

        # Display both components
        self.play(
            Create(cdna),
            Write(cdna_labels),
            run_time=0.7
        )
        self.wait(1.5)
        self.play(
            Create(aav_vector),
            Write(aav_label),
            run_time=0.7
        )
        self.wait(2)

        # 3. Move cDNA into the viral vector
        # Scale down cDNA to fit inside
        self.play(
            cdna.animate.scale(0.4).move_to(aav_vector.get_center()),
            FadeOut(cdna_labels),
            run_time=1
        )
        self.wait(0.5)

        # Group the combination
        aav_with_cdna = VGroup(aav_vector, cdna)
```

```
        # 4. Zoom and move to center, transform label
        aav_new_label = Tex("\\textbf{AAV2 serotype}", font_size=32, color=PURPLE)
        aav_new_label.move_to(DOWN * 2.5)

        self.play(
            aav_with_cdna.animate.scale(1.5).move_to(ORIGIN),
            Transform(aav_label, aav_new_label),
            run_time=1
        )
        self.wait(3)

class Scene7_LuxturnaInjection(Scene):
    """Luxturna Injection Animation"""
    def construct(self):
        scene_title = Tex("{\\bfseries Luxturna Mechanism}", font_size=48, color=YELLOW)
        scene_title.move_to(UP * 3)
        self.add(scene_title)

        # 1. Start from where Scene 6 ended - recreate the AAV2 with cDNA at center
        aav_capsid = RegularPolygon(n=6, radius=1.2, color=PURPLE, stroke_width=4, fill_color=PURPLE,
        ↪  fill_opacity=0.2)

        spike_positions = [
            UP * 1.2,
            UP * 0.6 + RIGHT * 1.0,
            DOWN * 0.6 + RIGHT * 1.0,
            DOWN * 1.2,
            DOWN * 0.6 + LEFT * 1.0,
            UP * 0.6 + LEFT * 1.0
        ]

        spikes = VGroup()
        for pos in spike_positions:
            spike = Triangle(color=PURPLE, fill_color=PURPLE, fill_opacity=0.8)
            spike.scale(0.15)
            spike.move_to(pos)
            angle = np.arctan2(pos[1], pos[0])
            spike.rotate(angle + PI/2)
            spikes.add(spike)

        aav_vector = VGroup(aav_capsid, spikes)

        cdna = ParametricFunction(
            lambda t: np.array([
                0.2 * np.sin(4 * PI * t),
                2 * t - 1,
                0
            ]),
            t_range=[0, 1],
            color=GREEN,
            stroke_width=6
        ).scale(0.4)

        aav_with_cdna = VGroup(aav_vector, cdna).scale(1.5).move_to(ORIGIN)
        aav_label = Tex("\\textbf{AAV2 serotype}", font_size=32, color=PURPLE).move_to(DOWN * 2.5)

        self.add(aav_with_cdna, aav_label)

        # 2. Zoom out AAV2 and move to left, create impaired cell on right-center
        self.play(
            aav_with_cdna.animate.scale(0.5).move_to(LEFT * 4),
            FadeOut(aav_label),
            run_time=1
```

```
        )

        # Create cell with impaired DNA (red) in nucleus
        cell = Circle(radius=2, color=BLUE, stroke_width=3, fill_color=BLUE, fill_opacity=0.1)
        nucleus = Circle(radius=0.8, color=PURPLE, stroke_width=3, fill_color=PURPLE,
        ↪  fill_opacity=0.15)

        # Impaired DNA (red)
        impaired_dna = ParametricFunction(
            lambda t: np.array([
                0.2 * np.sin(4 * PI * t),
                0.6 * t - 0.3,
                0
            ]),
            t_range=[0, 1],
            color=RED,
            stroke_width=4
        )

        cell_group = VGroup(cell, nucleus, impaired_dna)
        cell_group.move_to(RIGHT * 1.5)

        # Add cell label for impaired state
        cell_label = Tex("\\textbf{Cell (impaired DNA)}", font_size=32, color=RED)
        cell_label.next_to(cell_group, DOWN, buff=0.3)

        self.play(Create(cell_group), Write(cell_label), run_time=1)

        # 3. Vector moves towards cell and releases cDNA
        self.play(
            aav_with_cdna.animate.move_to(cell.get_left() + LEFT * 0.5),
            run_time=1.2
        )

        # Extract and scale up the cDNA for delivery
        cdna_delivery = cdna.copy().scale(2)

        # Create fixed cell label
        fixed_cell_label = Tex("\\textbf{Cell (fixed DNA)}", font_size=32, color=GREEN)
        fixed_cell_label.next_to(cell_group, DOWN, buff=0.3)

        self.play(
            cdna_delivery.animate.move_to(nucleus.get_center()),
            impaired_dna.animate.set_color(GREEN),
            Transform(cell_label, fixed_cell_label),
            run_time=1.5
        )

        # Remove the delivery cDNA and keep the transformed one
        self.remove(cdna_delivery)

        # 4. Vector fades out, cell moves to center and zooms slightly
        self.play(FadeOut(aav_with_cdna), run_time=0.5)

        self.play(
            cell_group.animate.scale(0.9).move_to(ORIGIN),
            cell_label.animate.next_to(ORIGIN + DOWN * 2, DOWN, buff=0),
            run_time=1
        )

        # 5. Cell produces functional retinoid isomerase (blue hexagons)
        enzyme_label = Tex("\\textbf{Functional Retinoid\\\\Isomerase (RPE65)}", font_size=32,
        ↪  color=BLUE)
        enzyme_label.move_to(DOWN * 3)
```

```
        # Create enzymes with fixed random positions
        enzyme_positions = [
            UP * 2 + LEFT * 5,
            UP * 1.5 + RIGHT * 5.5,
            UP * 0.5 + LEFT * 6,
            DOWN * 0.8 + RIGHT * 6,
            DOWN * 2 + LEFT * 5.5,
            DOWN * 2.5 + RIGHT * 4.5,
            UP * 2.8 + LEFT * 2,
            UP * 2.5 + RIGHT * 3,
            DOWN * 3 + LEFT * 3,
            DOWN * 2.8 + RIGHT * 2.5,
            UP * 1 + LEFT * 3.5,
            DOWN * 1.5 + LEFT * 4
        ]

        enzymes = VGroup()
        animations = []

        for i, pos in enumerate(enzyme_positions):
            enzyme = RegularPolygon(n=6, radius=0.15, color=BLUE, fill_color=BLUE, fill_opacity=0.7)
            enzyme.move_to(cell.get_center())
            enzymes.add(enzyme)

            # Stagger the animations slightly
            animations.append(
                AnimationGroup(
                    enzyme.animate.move_to(pos),
                    lag_ratio=0.1
                )
            )

        self.add(enzymes)

        # Animate enzymes leaving the cell
        self.play(
            LaggedStart(*[enzyme.animate.move_to(pos) for enzyme, pos in zip(enzymes,
            ↪  enzyme_positions)],
                        lag_ratio=0.08),
            Write(enzyme_label),
            run_time=2.8
        )

        self.wait(2)

class Scene8_ClinicalResultsEyes(Scene):
    """Clinical Results Text"""
    def construct(self):
        # Title
        title = Tex("{\\bfseries Clinical Results}", font_size=56, color=YELLOW)
        title.move_to(UP * 2.5)

        # Result statements
        result1 = Tex("$\\bullet$ Significant improvements in retinal function", font_size=32)
        result2 = Tex("$\\bullet$ Increased light sensitivity", font_size=32)
        result3 = Tex("$\\bullet$ Enhanced pupillary responses", font_size=32)
        result4 = Tex("$\\bullet$ Improved vision in dim environments", font_size=32, color=GREEN)

        results = VGroup(result1, result2, result3, result4).arrange(DOWN, buff=0.5,
        ↪  aligned_edge=LEFT)
        results.move_to(DOWN * 0.5)

        # Animate title
```

```
        self.play(Write(title), run_time=1)
        self.wait(0.5)

        # Animate results appearing one by one
        self.play(FadeIn(result1, shift=UP * 0.3), run_time=0.8)
        self.wait(0.3)
        self.play(FadeIn(result2, shift=UP * 0.3), run_time=0.8)
        self.wait(0.3)
        self.play(FadeIn(result3, shift=UP * 0.3), run_time=0.8)
        self.wait(0.3)
        self.play(FadeIn(result4, shift=UP * 0.3), run_time=0.8)

        self.wait(3)
```

---

application_blood_disorder.py

This file contains code for creating animations about gene therapy applications for blood disorders, specifically CRISPR treatment for sickle cell disease.

```
from manim import *

# BLOOD DISORDER APPLICATION SCENES

class Scene9_Chapter2Title(Scene):
    """Chapter 2 Title: Gene Therapy for Blood Disorders"""
    def construct(self):
        # Text animation
        chapter = Tex("{\\scshape\\bfseries CHAPTER II}", font_size=50, color=YELLOW)
        title = Tex("{\\bfseries Blood Disorders}", font_size=64, color=YELLOW)
        separator = Line(start=LEFT * 3, end=RIGHT * 3, color=ORANGE, stroke_width=4)

        chapter.move_to(UP * 0.75)
        title.move_to(DOWN * 0.75)
        separator.move_to(ORIGIN)

        self.play(
            Write(chapter, run_time=1.5),
            Create(separator, run_time=1.5),
            Write(title, run_time=1.5)
        )

        self.wait(3)

class Scene10_SickleCellDiseaseTitle(Scene):
    """Sickle Cell Disease Title Scene"""
    def construct(self):
        # Text animation
        title = Tex("{\\bfseries Sickle Cell Disease}", font_size=72, color=YELLOW)

        self.play(Write(title, run_time=1))

        self.wait(3)

class Scene11_CRISPRTreatment(Scene):
    """CRISPR-Cas9 Treatment for Sickle Cell Disease"""
    def construct(self):
        # Part 1: Show BCL11A gene suppressing fetal hemoglobin
        bcl11a_gene = Rectangle(width=2, height=0.8, color=RED, fill_color=RED, fill_opacity=0.3)
        bcl11a_label = Tex("\\textbf{BCL11A\\\\gene}", font_size=24, color=RED)
        bcl11a_label.next_to(bcl11a_gene, UP, buff=0.3)

        bcl11a_group = VGroup(bcl11a_gene, bcl11a_label)
```

```
        bcl11a_group.move_to(UP * 2 + LEFT * 3)

        # Fetal hemoglobin (suppressed, shown as small gray circles)
        fetal_hb_suppressed = VGroup(*[
            Circle(radius=0.15, color=GRAY, fill_color=GRAY, fill_opacity=0.4)
            for _ in range(3)
        ]).arrange(RIGHT, buff=0.3)
        fetal_hb_suppressed.move_to(DOWN * 1 + LEFT * 3)

        fetal_label = Tex("\\textbf{Fetal Hb\\\\(suppressed)}", font_size=20, color=GRAY)
        fetal_label.next_to(fetal_hb_suppressed, DOWN, buff=0.3)

        # Suppression arrow
        suppression_arrow = Arrow(
            start=bcl11a_gene.get_bottom(),
            end=fetal_hb_suppressed.get_top(),
            color=RED,
            stroke_width=6
        )

        self.play(
            Create(bcl11a_group),
            Create(suppression_arrow),
            Create(fetal_hb_suppressed),
            Write(fetal_label),
            run_time=1.5
        )
        self.wait(1)

        # Part 2: Show CRISPR-Cas9 coming in to edit BCL11A
        crispr_cas9 = VGroup(
            Rectangle(width=1.5, height=0.6, color=BLUE, fill_color=BLUE, fill_opacity=0.5),
            Tex("\\textbf{CRISPR-\\\\Cas9}", font_size=20, color=BLUE)
        )
        crispr_cas9[1].move_to(crispr_cas9[0].get_center())
        crispr_cas9.move_to(UP * 2 + RIGHT * 2)

        self.play(Create(crispr_cas9), run_time=1)
        self.wait(0.5)

        # CRISPR moves to BCL11A
        self.play(crispr_cas9.animate.move_to(bcl11a_gene.get_center()), run_time=1.5)

        # Flash effect showing gene editing
        flash = Flash(bcl11a_gene.get_center(), color=YELLOW, flash_radius=1)
        self.play(flash)

        # BCL11A becomes deactivated (crossed out)
        cross_line1 = Line(
            start=bcl11a_gene.get_corner(UL),
            end=bcl11a_gene.get_corner(DR),
            color=YELLOW,
            stroke_width=6
        )
        cross_line2 = Line(
            start=bcl11a_gene.get_corner(UR),
            end=bcl11a_gene.get_corner(DL),
            color=YELLOW,
            stroke_width=6
        )

        deactivated_label = Tex("\\textbf{BCL11A\\\\(deactivated)}", font_size=24, color=GRAY)
        deactivated_label.move_to(bcl11a_label.get_center())
```

```
        self.play(
            Create(cross_line1),
            Create(cross_line2),
            Transform(bcl11a_label, deactivated_label),
            FadeOut(crispr_cas9),
            suppression_arrow.animate.set_color(GRAY).set_opacity(0.3),
            run_time=1.5
        )
        self.wait(1)

        # Part 3: Fetal hemoglobin production resumes
        fetal_hb_active = VGroup(*[
            Circle(radius=0.2, color=GREEN, fill_color=GREEN, fill_opacity=0.7)
            for _ in range(6)
        ]).arrange_in_grid(rows=2, cols=3, buff=0.3)
        fetal_hb_active.move_to(fetal_hb_suppressed.get_center() + DOWN * 0.5)

        active_label = Tex("\\textbf{Fetal Hb\\\\(high-level)}", font_size=20, color=GREEN)
        active_label.next_to(fetal_hb_active, DOWN, buff=0.3)

        self.play(
            Transform(fetal_hb_suppressed, fetal_hb_active),
            Transform(fetal_label, active_label),
            run_time=1.5
        )
        self.wait(1)

        # Part 4: Show effect on red blood cells
        # Fade everything out
        self.play(
            FadeOut(bcl11a_group),
            FadeOut(cross_line1),
            FadeOut(cross_line2),
            FadeOut(suppression_arrow),
            FadeOut(fetal_hb_suppressed),
            FadeOut(fetal_label),
            run_time=1
        )

        # 1. Create sickle cells and normal hemoglobin scattered across screen
        # Sickle cells (red crescents)
        sickle_positions = [
            UP * 2.5 + LEFT * 4,
            UP * 1.5 + LEFT * 1,
            UP * 0.5 + LEFT * 5,
            DOWN * 0.5 + LEFT * 2.5,
            DOWN * 1.5 + LEFT * 5.5,
            UP * 2 + RIGHT * 1,
            DOWN * 2 + LEFT * 0.5,
        ]

        sickle_cells = VGroup()
        for pos in sickle_positions:
            arc1 = Arc(radius=0.3, angle=PI, color=RED, stroke_width=3, fill_color=RED,
            ↪  fill_opacity=0.4)
            arc2 = Arc(radius=0.25, angle=-PI, color=RED, stroke_width=3, fill_color=RED,
            ↪  fill_opacity=0.4).next_to(arc1, RIGHT, buff=0)
            sickle = VGroup(arc1, arc2).rotate(PI/4)
            sickle.move_to(pos)
            sickle_cells.add(sickle)

        # Normal hemoglobin (green circles)
        normal_hb_positions = [
```

```
            UP * 3 + LEFT * 2,
            UP * 1 + RIGHT * 4,
            DOWN * 1 + RIGHT * 5,
            DOWN * 2.5 + RIGHT * 2,
            UP * 0.5 + RIGHT * 2,
        ]

        normal_hbs = VGroup()
        for pos in normal_hb_positions:
            normal_hb = Circle(radius=0.25, color=GREEN, fill_color=GREEN, fill_opacity=0.5, stroke_width=3)
            normal_hb.move_to(pos)
            normal_hbs.add(normal_hb)

        # Legend at bottom left
        legend_box = Rectangle(width=3.5, height=1.8, color=WHITE, stroke_width=2)
        legend_box.move_to(DOWN * 2.7 + LEFT * 4.5)

        # Legend items
        sickle_legend_icon = VGroup(
            Arc(radius=0.15, angle=PI, color=RED, stroke_width=2, fill_color=RED, fill_opacity=0.4),
        )
        sickle_legend_icon[0].next_to(sickle_legend_icon[0], RIGHT, buff=0)
        sickle_legend_label = Tex("Sickle Hb", font_size=18, color=RED)
        sickle_legend = VGroup(sickle_legend_icon, sickle_legend_label).arrange(RIGHT, buff=0.3)

        normal_legend_icon = Circle(radius=0.15, color=GREEN, fill_color=GREEN, fill_opacity=0.5, stroke_width=2)
        normal_legend_label = Tex("Normal Hb", font_size=18, color=GREEN)
        normal_legend = VGroup(normal_legend_icon, normal_legend_label).arrange(RIGHT, buff=0.3)

        fetal_legend_icon = Circle(radius=0.15, color=BLUE, fill_color=BLUE, fill_opacity=0.5, stroke_width=2)
        fetal_legend_label = Tex("Fetal Hb", font_size=18, color=BLUE)
        fetal_legend = VGroup(fetal_legend_icon, fetal_legend_label).arrange(RIGHT, buff=0.3)

        legend_items = VGroup(sickle_legend, normal_legend, fetal_legend).arrange(DOWN, aligned_edge=LEFT, buff=0.2)
        legend_items.move_to(legend_box.get_center())

        legend_group = VGroup(legend_box, legend_items)

        # 2. Counter at bottom right showing functional hemoglobin percentage
        counter_box = Rectangle(width=3, height=1.2, color=WHITE, stroke_width=2)
        counter_box.move_to(DOWN * 2.9 + RIGHT * 4.5)

        counter_label = Tex("\\textbf{Functional Hb}", font_size=20)
        counter_label.move_to(counter_box.get_center() + UP * 0.3)

        initial_percentage = int((len(normal_hbs) / (len(sickle_cells) + len(normal_hbs))) * 100)
        counter_value = Tex(f"\\textbf{{{initial_percentage}\\%}}", font_size=32, color=YELLOW)
        counter_value.move_to(counter_box.get_center() + DOWN * 0.25)

        counter_group = VGroup(counter_box, counter_label, counter_value)

        # Fade in everything
        self.play(
            Create(sickle_cells),
            Create(normal_hbs),
            Create(legend_group),
            Create(counter_group),
            run_time=1.5
        )
        self.wait(1)
```

```
        # 3. Fade in fetal hemoglobin (blue circles) and update counter
        fetal_hb_positions = [
            UP * 2.8 + RIGHT * 3.5,
            UP * 0.8 + LEFT * 3.5,
            UP * 1.8 + RIGHT * 5.5,
            DOWN * 0.3 + RIGHT * 0.5,
            DOWN * 1.8 + RIGHT * 4.5,
            DOWN * 2.5 + LEFT * 3,
            UP * 3.2 + LEFT * 0.5,
            DOWN * 0.8 + LEFT * 4.5,
        ]

        fetal_hbs = VGroup()
        for pos in fetal_hb_positions:
            fetal_hb = Circle(radius=0.25, color=BLUE, fill_color=BLUE, fill_opacity=0.5,
            ↪  stroke_width=3)
            fetal_hb.move_to(pos)
            fetal_hbs.add(fetal_hb)

        # Calculate new percentage
        new_percentage = int((len(normal_hbs) + len(fetal_hbs)) / (len(sickle_cells) + len(normal_hbs)
        ↪  + len(fetal_hbs)) * 100)
        new_counter_value = Tex(f"\\textbf{{{new_percentage}\\%}}", font_size=32, color=GREEN)
        new_counter_value.move_to(counter_value.get_center())

        self.play(
            FadeIn(fetal_hbs, lag_ratio=0.1),
            Transform(counter_value, new_counter_value),
            run_time=2
        )
        self.wait(1.5)

        # Final message
        cure_text = Tex("\\textbf{Malfunctioned Haemoglobin Diluted}", font_size=40, color=WHITE)
        cure_text_box = Rectangle(width=8, height=1, color=GREEN, stroke_width=3, fill_color=GREEN,
        ↪  fill_opacity=0.3)
        cure_text.move_to(ORIGIN)

        self.play(Create(cure_text_box), Write(cure_text), run_time=1)
        self.wait(2)
```

application_blood_cancer.py

This file contains code for creating animations about gene therapy applications for blood cancer, specifically CAR-T cell therapy for leukemia and its side effects.

```
from manim import *
import random

# BLOOD CANCER APPLICATION SCENES

class Scene12_Chapter3Title(Scene):
    """Chapter 3 Title: Leukaemia and Lymphoma"""
    def construct(self):
        # Text animation
        chapter = Tex("{\\scshape\\bfseries CHAPTER III}", font_size=50, color=YELLOW)
        title = Tex("{\\bfseries Leukaemia and Lymphoma}", font_size=64, color=YELLOW)
        separator = Line(start=LEFT * 3, end=RIGHT * 3, color=ORANGE, stroke_width=4)

        chapter.move_to(UP * 0.75)
        title.move_to(DOWN * 0.75)
```

```
        separator.move_to(ORIGIN)

        self.wait(1) # for editing

        self.play(
            Write(chapter, run_time=1.5),
            Create(separator, run_time=1.5),
            Write(title, run_time=1.5)
        )

        self.wait(3)

class Scene13_WBCDivision(Scene):
    """White Blood Cell Division and Mutation Introduction Scene"""
    def construct(self):
        # Create initial white blood cell
        initial_cell = Circle(radius=0.3, color=WHITE, fill_opacity=0.8, fill_color=WHITE,
        ↪  stroke_width=2)
        initial_cell.move_to(ORIGIN)

        # Store all cells in a list
        cells = [initial_cell]

        self.add(initial_cell)
        self.wait(1)

        # Division loop
        max_cells = 100
        division_interval = 0.15  # Time between divisions
        cell_radius = 0.3

        def find_non_overlapping_position(parent_pos, existing_cells, max_attempts=50):
            """Find a position for new cell that doesn't overlap with existing cells"""
            for attempt in range(max_attempts):
                angle = np.random.uniform(0, 2 * np.pi)
                distance = cell_radius * 2.5  # Start at safe distance
                offset = np.array([np.cos(angle), np.sin(angle), 0]) * distance
                new_pos = parent_pos + offset

                # Check if position is valid (not overlapping and within frame)
                if abs(new_pos[0]) < config.frame_width / 2 - cell_radius and \
                   abs(new_pos[1]) < config.frame_height / 2 - cell_radius:

                    # Check overlap with existing cells
                    overlapping = False
                    for cell in existing_cells:
                        dist = np.linalg.norm(new_pos - cell.get_center())
                        if dist < cell_radius * 2.2:  # Minimum safe distance
                            overlapping = True
                            break

                    if not overlapping:
                        return new_pos

            # If no good position found, return random position with larger offset
            angle = np.random.uniform(0, 2 * np.pi)
            distance = cell_radius * 4
            offset = np.array([np.cos(angle), np.sin(angle), 0]) * distance
            return parent_pos + offset

        while len(cells) < max_cells:
            # Select a random cell to divide from all existing cells
            if len(cells) > 0:
                parent = random.choice(cells)
```

```
                parent_pos = parent.get_center()

                # Find positions for two daughter cells
                pos1 = find_non_overlapping_position(parent_pos, cells)

                # Create first daughter cell
                new_cell1 = Circle(radius=cell_radius, color=WHITE, fill_opacity=0.8,
                ↪  fill_color=WHITE, stroke_width=2)
                new_cell1.move_to(parent_pos)

                # Temporarily add to list to check for second position
                temp_cells = cells + [new_cell1]
                pos2 = find_non_overlapping_position(parent_pos, temp_cells)

                # Create second daughter cell
                new_cell2 = Circle(radius=cell_radius, color=WHITE, fill_opacity=0.8,
                ↪  fill_color=WHITE, stroke_width=2)
                new_cell2.move_to(parent_pos)

                # Remove parent cell and add daughter cells
                self.remove(parent)
                cells.remove(parent)

                cells.append(new_cell1)
                cells.append(new_cell2)

                self.add(new_cell1, new_cell2)

                # Animate cells moving to their positions
                self.play(
                    new_cell1.animate.move_to(pos1),
                    new_cell2.animate.move_to(pos2),
                    run_time=0.3,
                    rate_func=smooth
                )

                if len(cells) >= max_cells:
                    break

        self.wait(2)

class Scene14_CARTCellTherapy(Scene):
    """CAR T-cell Therapy Process"""
    def construct(self):
        # Text animation
        title = Tex("{\\bfseries CAR-T Cell Therapy}", font_size=72, color=YELLOW)

        self.wait(2)  # for editing

        self.play(Write(title, run_time=1))

        self.wait(3)

class Scene15_HighRateBut(Scene):
    def construct(self):
        # Text animation
        text = Tex("\\textbf{High Remission Rate}", font_size=64, color=YELLOW)
        text.move_to(UP * 0.5)
        self.wait(1.5)
        self.play(Write(text, run_time=1))
        self.wait(1)

        but_text = Tex("\\textbf{But...?}", font_size=64, color=RED)
        but_text.move_to(DOWN * 0.5)
```

```
        self.play(Write(but_text, run_time=1))
        self.wait(2)

class Scene16_CytokineStorm(Scene):
    """Cytokine Storm Explanation Scene"""
    def construct(self):
        # 1. T cell
        t_cell = Circle(radius=0.5, color=BLUE, fill_color=BLUE, fill_opacity=0.5, stroke_width=3)
        t_cell.move_to(UP * 2.5)
        t_cell_label = Tex("\\textbf{T Cell}", font_size=24, color=BLUE).next_to(t_cell, DOWN,
        ↪  buff=0.3)
        t_cell_group = VGroup(t_cell, t_cell_label)

        # 2. Cancer Cell
        cancer_cell = Circle(radius=0.5, color=RED, fill_color=RED, fill_opacity=0.5, stroke_width=3)
        cancer_cell.move_to(DOWN * 2.5 + LEFT * 3)
        cancer_cell_label = Tex("\\textbf{Cancer Cell}", font_size=24, color=RED).next_to(cancer_cell,
        ↪  DOWN, buff=0.3)
        cancer_cell_group = VGroup(cancer_cell, cancer_cell_label)

        # 3. Healthy Cell
        healthy_cell = Circle(radius=0.5, color=GREEN, fill_color=GREEN, fill_opacity=0.5,
        ↪  stroke_width=3)
        healthy_cell.move_to(DOWN * 2.5 + RIGHT * 3)
        healthy_cell_label = Tex("\\textbf{Healthy Cell}", font_size=24,
        ↪  color=GREEN).next_to(healthy_cell, DOWN, buff=0.3)
        healthy_cell_group = VGroup(healthy_cell, healthy_cell_label)

        # arrows
        arrow_to_cancer = Arrow(
            start=t_cell_group.get_bottom(),
            end=cancer_cell_group.get_top() + DOWN * 0.1,
            color=YELLOW,
            buff=0.5,
            stroke_width=6
        )

        arrow_to_healthy = Arrow(
            start=t_cell_group.get_bottom(),
            end=healthy_cell_group.get_top() + DOWN * 0.1,
            color=YELLOW,
            buff=0.5,
            stroke_width=6
        )

        # text
        text = Tex("\\textbf{Cytokine Storm}", font_size=48, color=ORANGE)
        text.next_to(arrow_to_healthy.get_center(), RIGHT, buff=1)

        self.wait(1)
        self.play(
            Create(t_cell), Write(t_cell_label),
            Create(cancer_cell), Write(cancer_cell_label),
            Create(healthy_cell), Write(healthy_cell_label),
            run_time=0.5
        )
        self.wait(1)
        self.play(Create(arrow_to_cancer), run_time=0.5)
        self.play(Create(arrow_to_healthy), Write(text), run_time=0.5)

        self.wait(3)
```

application_neuromuscular.py

This file contains code for creating animations about gene therapy applications for neuromuscular diseases, specifically Nusinersen therapy for Spinal Muscular Atrophy (SMA).

```python
from manim import *

# NEUROMUSCULAR DISEASE APPLICATION SCENES

class Scene17_Chapter4Title(Scene):
    def construct(self):
        # Text animation
        chapter = Tex("{\\scshape\\bfseries CHAPTER IV}", font_size=50, color=YELLOW)
        title = Tex("{\\bfseries Spinal Muscular Atrophy}", font_size=64, color=YELLOW)
        separator = Line(start=LEFT * 3, end=RIGHT * 3, color=ORANGE, stroke_width=4)

        chapter.move_to(UP * 0.75)
        title.move_to(DOWN * 0.75)
        separator.move_to(ORIGIN)

        self.play(
            Write(chapter, run_time=1.5),
            Create(separator, run_time=1.5),
            Write(title, run_time=1.5)
        )

        self.wait(3)

class Scene18_BackupSMN2Gene(Scene):
    def construct(self):
        # title
        title = Tex("\\textbf{SMN2 Gene As a Backup}", font_size=48, color=YELLOW)
        title.move_to(UP * 3)
        self.play(Write(title), run_time=1)
        self.wait(1)

        # A segment of SMN2 RNA, represented by two parallel wavy lines (sine waves)
        x_start = -6
        x_end = 6
        amplitude = 0.3
        frequency = 1
        wave_func = lambda x: amplitude * np.sin(frequency * x)
        smn2_rna = VGroup(
            ParametricFunction(
                lambda t: np.array([t, wave_func(t) + 0.3, 0]),
                t_range=[x_start, x_end],
                color=BLUE,
                stroke_width=4
            ),
            ParametricFunction(
                lambda t: np.array([t, wave_func(t) - 0.3, 0]),
                t_range=[x_start, x_end],
                color=BLUE,
                stroke_width=4
            )
        )
        smn2_rna.scale(1.3)

        smn2_label = Tex("\\textbf{SMN2 RNA}", font_size=32, color=BLUE).next_to(smn2_rna, UP,
        ↪  buff=0.5)
        smn2_label.shift(LEFT * 1.7 + DOWN * 1)

        self.add(smn2_rna)
```

```
        self.play(Write(smn2_label), run_time=1)
        self.wait(1)

        # An error marker (red exclamation mark, with a circle around) pops on the RNA
        exclamation_mark = Tex("!", font_size=56, color=RED)
        error_circle = Circle(radius=0.3, color=RED, stroke_width=3)
        error_marker = VGroup(error_circle, exclamation_mark).move_to(smn2_rna.get_center() + RIGHT *
        ↪  2 + UP * 0.35)
        error_flash = Flash(error_marker.get_center(), color=YELLOW, flash_radius=0.4,
        ↪  line_stroke_width=6)

        self.play(Create(error_marker), error_flash, run_time=1)
        self.wait(1)

        # two dashed lines to indicate splicing
        splice_line1 = DashedLine(
            start=smn2_rna[0].point_from_proportion(0.5),
            end=smn2_rna[1].point_from_proportion(0.5),
            color=WHITE,
            stroke_width=4,
            dash_length=0.06,
            buff=0
        )
        splice_line2 = DashedLine(
            start=smn2_rna[0].point_from_proportion(0.8),
            end=smn2_rna[1].point_from_proportion(0.8),
            color=WHITE,
            stroke_width=4,
            dash_length=0.06,
            buff=0
        )
        exon7_text = Tex("\\textbf{Exon 7} \\\\excluded", font_size=28,
        ↪  color=RED).next_to(splice_line2, RIGHT, buff=0.3)
        exon7_text.next_to(error_marker, DOWN, buff=0.2)
        self.play(Create(splice_line1), Create(splice_line2), Write(exon7_text), run_time=1)
        self.wait(1)

        # arrow down
        arrow = Arrow(
            start=splice_line1.get_bottom() + DOWN * 0.2,
            end=splice_line1.get_bottom() + DOWN * 1.2,
            color=YELLOW,
            buff=0,
            stroke_width=6
        )

        # 3/4 circle to represent truncated protein
        truncated_protein = Arc(
            radius=0.4,
            start_angle=PI / 4,
            angle=3 * PI / 2,
            color=ORANGE,
            stroke_width=6
        ).move_to(arrow.get_end() + DOWN * 0.5)
        truncated_label = Tex("\\textbf{Truncated SMN Protein}", font_size=28,
        ↪  color=ORANGE).next_to(truncated_protein, DOWN, buff=0.3)

        self.play(
            Create(arrow),
            Write(truncated_label),
            Create(truncated_protein),
            run_time=1
        )
        self.wait(1)
```

```python

class Scene19_NusinersenTherapy(Scene):
    def construct(self):
        # recreate the ending state of Scene 18
        # title
        title = Tex("\\textbf{SMN2 Gene As a Backup}", font_size=48, color=YELLOW)
        title.move_to(UP * 3)
        self.add(title)

        # SMN2 RNA
        x_start = -6
        x_end = 6
        amplitude = 0.3
        frequency = 1
        wave_func = lambda x: amplitude * np.sin(frequency * x)
        smn2_rna = VGroup(
            ParametricFunction(
                lambda t: np.array([t, wave_func(t) + 0.3, 0]),
                t_range=[x_start, x_end],
                color=BLUE,
                stroke_width=4
            ),
            ParametricFunction(
                lambda t: np.array([t, wave_func(t) - 0.3, 0]),
                t_range=[x_start, x_end],
                color=BLUE,
                stroke_width=4
            )
        )
        smn2_rna.scale(1.3)
        smn2_label = Tex("\\textbf{SMN2 RNA}", font_size=32, color=BLUE).next_to(smn2_rna, UP,
        ↪  buff=0.5)
        smn2_label.shift(LEFT * 1.7 + DOWN * 1)
        self.add(smn2_rna, smn2_label)

        # error marker
        exclamation_mark = Tex("!", font_size=56, color=RED)
        error_circle = Circle(radius=0.3, color=RED, stroke_width=3)
        error_marker = VGroup(error_circle, exclamation_mark).move_to(smn2_rna.get_center() + RIGHT *
        ↪  2 + UP * 0.35)
        self.add(error_marker)

        # splice lines
        splice_line1 = DashedLine(
            start=smn2_rna[0].point_from_proportion(0.5),
            end=smn2_rna[1].point_from_proportion(0.5),
            color=WHITE,
            stroke_width=4,
            dash_length=0.06,
            buff=0
        )
        splice_line2 = DashedLine(
            start=smn2_rna[0].point_from_proportion(0.8),
            end=smn2_rna[1].point_from_proportion(0.8),
            color=WHITE,
            stroke_width=4,
            dash_length=0.06,
            buff=0
        )
        exon7_text = Tex("\\textbf{Exon 7} \\\\excluded", font_size=28,
        ↪  color=RED).next_to(splice_line2, RIGHT, buff=0.3)
        exon7_text.next_to(error_marker, DOWN, buff=0.2)
        self.add(splice_line1, splice_line2, exon7_text)

```

```
        # arrow
        arrow = Arrow(
            start=splice_line1.get_bottom() + DOWN * 0.2,
            end=splice_line1.get_bottom() + DOWN * 1.2,
            color=YELLOW,
            buff=0,
            stroke_width=6
        )
        self.add(arrow)

        # truncated protein
        truncated_protein = Arc(
            radius=0.4,
            start_angle=PI / 4,
            angle=3 * PI / 2,
            color=ORANGE,
            stroke_width=6
        ).move_to(arrow.get_end() + DOWN * 0.5)
        truncated_label = Tex("\\textbf{Truncated SMN Protein}", font_size=28,
        ↪  color=ORANGE).next_to(truncated_protein, DOWN, buff=0.3)
        self.add(truncated_protein, truncated_label)

        self.wait(1)

        #### NEW SCENE

        new_title = Tex("\\textbf{Nusinersen Therapy}", font_size=48, color=YELLOW)
        new_title.move_to(UP * 3)
        self.play(
            Transform(title, new_title),
            FadeOut(exon7_text),
            FadeOut(splice_line1), FadeOut(splice_line2),
            run_time=1
        )

        # Proofreader (a purple oval)
        proofreader = Ellipse(width=1.8, height=1.3, color=PURPLE, fill_color=PURPLE,
        ↪  fill_opacity=0.5, stroke_width=3)
        proofreader_label = Tex("\\textbf{Nusinersen}", font_size=32, color=WHITE)
        proofreader_label.move_to(proofreader.get_center())
        proofreader_group = VGroup(proofreader, proofreader_label)
        # move out of screen on the right
        proofreader_group.move_to(error_marker.get_center() + RIGHT * 8 + UP * 0.5)

        # animate proofreader moving to the error site
        self.play(
            proofreader_group.animate.move_to(error_marker.get_center() + UP * 0.5),
            run_time=1
        )
        # animate proofreader fixing the error
        self.play(
            FadeOut(error_marker),
            Flash(error_marker.get_center(), color=GREEN, flash_radius=0.4, line_stroke_width=6),
            run_time=1
        )
        self.wait(1)

        # fixed protein
        fixed_protein = Circle(radius=0.4, color=GREEN, stroke_width=6)
        fixed_label = Tex("\\textbf{Full-length SMN Protein}", font_size=28, color=GREEN)
        fixed_protein.move_to(truncated_protein.get_center())
        fixed_label.move_to(truncated_label.get_center())

        # animate splicing and protein change
```

```
        self.play(
            Transform(truncated_protein, fixed_protein),
            Transform(truncated_label, fixed_label),
            run_time=1
        )

        self.wait(2)
```

---

ethics.py

This file contains code for creating animations about ethical considerations in gene therapy, including historical cases, public opinion, and regulatory concerns.

```
from manim import *
import random

# ETHICAL DISCUSSION SCENES

class Scene20_EthicalTitle(Scene):
    """Chapter Title: Ethical Considerations"""
    def construct(self):
        # Text animation
        chapter = Tex("{\\scshape\\bfseries CHAPTER V}", font_size=50, color=YELLOW)
        title = Tex("{\\bfseries Gene Therapy Ethics}", font_size=64, color=YELLOW)
        separator = Line(start=LEFT * 3, end=RIGHT * 3, color=ORANGE, stroke_width=4)

        chapter.move_to(UP * 0.75)
        title.move_to(DOWN * 0.75)
        separator.move_to(ORIGIN)

        self.play(
            Write(chapter, run_time=1.5),
            Create(separator, run_time=1.5),
            Write(title, run_time=1.5)
        )

        self.wait(3)

class Scene21_HumanWithQuestionMarks(Scene):
    def construct(self):
        # human svg
        human = SVGMobject("./human_outline.svg", fill_color=BLUE, fill_opacity=0.3,
        ↪  stroke_color=BLUE, stroke_width=2, should_center=True)
        human.scale(5)
        human.move_to(DOWN*1.5)
        self.play(Create(human), run_time=2)

        # Create question marks with different properties
        question_marks = []
        colors = [RED, YELLOW, GREEN, PURPLE, ORANGE, PINK, TEAL]
        positions = [
            DOWN * 2 + LEFT * 3,
            UP * 3 + RIGHT * 2,
            LEFT * 4 + UP * 1,
            RIGHT * 4 + UP * 0.5,
            DOWN * 2.5 + RIGHT * 4,
            LEFT * 3.5 + UP * 3,
            RIGHT * 3 + DOWN * 3.5,
            LEFT * 2 + UP * 1.5,
            RIGHT * 2.5 + UP * 2,
            LEFT * 5 + UP * 2.5,
            RIGHT * 5 + UP * 1.5,
```

```
            DOWN * 3.5 + LEFT * 1.5,
        ]

        # Predefined sizes for each question mark
        sizes = [65, 72, 48, 55, 78, 43, 69, 51, 76, 58, 62, 71]

        for i, (pos, color, size) in enumerate(zip(positions, colors * 2, sizes)):
            qmark = Tex("?", font_size=size, color=color)
            qmark.move_to(pos)
            question_marks.append(qmark)

        # Fade in question marks with lag
        self.play(
            LaggedStart(*[FadeIn(qmark) for qmark in question_marks],
                        lag_ratio=0.5,
                        run_time=6)
        )

        # text
        philosophical_text = Tex("$\\bullet$ Philosophical question?", font_size=50, color=WHITE)
        medical_safety_text = Tex("$\\bullet$ Medical safety!", font_size=50, color=WHITE)

        philosophical_text.move_to(DOWN * 2.6 + LEFT * 4)
        medical_safety_text.next_to(philosophical_text, DOWN, aligned_edge=LEFT, buff=0.2)

        self.play(Write(philosophical_text, run_time=1))
        self.wait(0.5)
        self.play(Write(medical_safety_text, run_time=1))
        self.wait(3)

class Scene22_JesseGelsingerCaseTimeline(Scene):
    def construct(self):
        # Title
        title = Tex("\\textbf{Jesse Gelsinger Case Timeline}", font_size=48, color=YELLOW)
        title.move_to(UP * 3.5)
        self.play(Write(title, run_time=0.5))
        self.wait(0.5)

        # Timeline arrow
        timeline = Arrow(start=LEFT * 6 + UP * 3, end=LEFT * 6 + DOWN * 4, color=YELLOW,
        ↪  stroke_width=4)
        self.play(Create(timeline), run_time=0.5)
        self.wait(0.5)

        # Events
        events = [
            ("{\\bfseries 18 Jun 1981:}", "Jesse Gelsinger born, later diagnosed with OTC
            ↪  deficiency"),
            ("{\\bfseries 13 Sept 1999:}", "At 18, Gelsinger injected with experimental gene
            ↪  therapy"),
            ("{\\bfseries 17 Sept 1999:}", "Gelsinger died from organ failure due to immune
            ↪  response"),
        ]

        for i, event_text in enumerate(events):
            event = VGroup(
                Tex(event_text[0], font_size=28, color=WHITE),
                Tex(event_text[1], font_size=32, color=WHITE)
            )
            event.arrange(DOWN, aligned_edge=LEFT)
            x_pos = LEFT * 6
            y_pos = UP * 2 + (DOWN * 4) * (i / (len(events) - 1))
            marker = Dot(point=x_pos + y_pos, color=RED, radius=0.1)
            event.next_to(marker, RIGHT, buff=0.3)
```

```
            self.play(Create(marker), Write(event, run_time=1))
            self.wait(0.5)

        self.wait(3)

class Scene23_BubbleBoy(Scene):
    def construct(self):
        # Create 4 stick figures in blue bubbles arranged in 2x2 grid
        stick_figures = []
        bubbles = []

        # Positions for 2x2 grid
        positions = [
            UP * 1.5 + LEFT * 2,   # Top left
            UP * 1.5 + RIGHT * 2,  # Top right
            DOWN * 1.5 + LEFT * 2, # Bottom left
            DOWN * 1.5 + RIGHT * 2 # Bottom right
        ]

        for pos in positions:
            # Create stick figure
            stick = VGroup(
                Circle(radius=0.15, color=WHITE, fill_opacity=1),  # Head
                Line(start=UP * 0.25, end=DOWN * 0.5, color=WHITE),   # Body
                Line(start=UP * 0.25, end=DOWN * 0.15 + LEFT * 0.25, color=WHITE),  # Left arm
                Line(start=UP * 0.25, end=DOWN * 0.15 + RIGHT * 0.25, color=WHITE), # Right arm
                Line(start=DOWN * 0.5, end=DOWN * 0.8 + LEFT * 0.2, color=WHITE),  # Left leg
                Line(start=DOWN * 0.5, end=DOWN * 0.8 + RIGHT * 0.2, color=WHITE), # Right leg
            )
            stick[0].move_to(UP * 0.4)
            stick.scale(0.8)
            stick.move_to(pos)

            # Create bubble
            bubble = Circle(radius=1.2, color=BLUE, stroke_width=4, fill_opacity=0.3, fill_color=BLUE)
            bubble.move_to(pos)

            stick_figures.append(stick)
            bubbles.append(bubble)

        # Show all stick figures and blue bubbles
        self.play(
            *[Create(bubble) for bubble in bubbles],
            *[Create(stick) for stick in stick_figures],
            run_time=1
        )
        self.wait(0.5)

        # Change colors: bottom right to red, others to green
        # Also create warning text for bottom right
        warning_text = Tex("Accidental activation\\\\ of leukaemia (25\\%)", font_size=32, color=RED)
        warning_text.next_to(bubbles[3], DOWN, buff=0.2)

        self.play(
            bubbles[3].animate.set_color(RED),  # Bottom right to red
            *[bubbles[i].animate.set_color(GREEN) for i in range(3)],  # Others to green
            Write(warning_text),
            run_time=1
        )

        self.wait(3)

class Scene24_ImmuneToGeneTherapy(Scene):
```

```
    """Scene showing immune response to gene therapy"""
    def construct(self):
        # 1. Create green human outline on the right half
        human = SVGMobject("./human_outline.svg", fill_color=GREEN, fill_opacity=0.3,
        ↪  stroke_color=GREEN, stroke_width=3, should_center=True)
        human.scale(3)
        human.move_to(RIGHT * 3)

        # Create AAV viral vector with RNA strand (upper-left)
        aav_capsid = RegularPolygon(n=6, radius=1.2, color=PURPLE, stroke_width=4, fill_color=PURPLE,
        ↪  fill_opacity=0.2)

        spike_positions = [
            UP * 1.2,
            UP * 0.6 + RIGHT * 1.0,
            DOWN * 0.6 + RIGHT * 1.0,
            DOWN * 1.2,
            DOWN * 0.6 + LEFT * 1.0,
            UP * 0.6 + LEFT * 1.0
        ]

        spikes = VGroup()
        for pos in spike_positions:
            spike = Triangle(color=PURPLE, fill_color=PURPLE, fill_opacity=0.8)
            spike.scale(0.15)
            spike.move_to(pos)
            angle = np.arctan2(pos[1], pos[0])
            spike.rotate(angle + PI/2)
            spikes.add(spike)

        # RNA strand inside
        rna_strand = ParametricFunction(
            lambda t: np.array([
                0.2 * np.sin(4 * PI * t),
                1.5 * t - 0.75,
                0
            ]),
            t_range=[0, 1],
            color=GREEN,
            stroke_width=6
        ).scale(0.3)

        aav_capsid.scale(0.5)
        spikes.scale(0.5)
        aav_vector = VGroup(aav_capsid, spikes, rna_strand)
        aav_vector.move_to(UP * 2 + LEFT * 4)

        # Create CRISPR-Cas9 protein (lower-left)
        crispr_cas9 = VGroup(
            Rectangle(width=1.5, height=0.6, color=BLUE, fill_color=BLUE, fill_opacity=0.5),
            Tex("\\textbf{CRISPR-\\\\Cas9}", font_size=20, color=BLUE)
        )
        crispr_cas9.scale(1.4)
        crispr_cas9[1].move_to(crispr_cas9[0].get_center())
        crispr_cas9.move_to(DOWN * 2 + LEFT * 4)

        # Display all elements
        self.play(
            Create(human),
            Create(aav_vector),
            Create(crispr_cas9),
            run_time=1
        )
        self.wait(1)
```

```
        # 2. Vector approaches human but is bounced back
        target_pos = human.get_left() + LEFT * 0.5
        self.play(
            aav_vector.animate.move_to(target_pos),
            run_time=1
        )
        # self.wait(0.3)

        # Bounce back with shake
        self.play(
            aav_vector.animate.shift(LEFT * 0.3),
            run_time=0.2
        )
        # self.play(
        #     aav_vector.animate.shift(LEFT * 1.5),
        #     run_time=0.8,
        #     rate_func=smooth
        # )
        self.wait(0.5)

        # 3. Red cross forms on vector and it turns gray
        cross_line1 = Line(
            start=UP * 0.5 + LEFT * 0.5,
            end=DOWN * 0.5 + RIGHT * 0.5,
            color=RED,
            stroke_width=8
        )
        cross_line2 = Line(
            start=UP * 0.5 + RIGHT * 0.5,
            end=DOWN * 0.5 + LEFT * 0.5,
            color=RED,
            stroke_width=8
        )
        red_cross_vector = VGroup(cross_line1, cross_line2)
        red_cross_vector.move_to(aav_vector.get_center())

        self.play(
            Create(red_cross_vector),
            aav_capsid.animate.set_color(GRAY),
            aav_capsid.animate.set_fill(GRAY, opacity=0.2),
            *[spike.animate.set_color(GRAY).set_fill(GRAY, opacity=0.2) for spike in spikes],
            aav_vector.animate.set_stroke(GRAY),
            rna_strand.animate.set_color(GRAY),
            run_time=1
        )
        self.wait(1)

        # 4. CRISPR-Cas9 enters the human
        self.play(
            crispr_cas9.animate.move_to(human.get_center()),
            run_time=2
        )
        self.wait(0.5)

        # Cross forms on human and human turns gray
        cross_line3 = Line(
            start=UP * 1.5 + LEFT * 1.5,
            end=DOWN * 1.5 + RIGHT * 1.5,
            color=RED,
            stroke_width=10
        )
        cross_line4 = Line(
```

```
            start=UP * 1.5 + RIGHT * 1.5,
            end=DOWN * 1.5 + LEFT * 1.5,
            color=RED,
            stroke_width=10
        )
        red_cross_human = VGroup(cross_line3, cross_line4)
        red_cross_human.move_to(human.get_center())

        self.play(
            Create(red_cross_human),
            human.animate.set_color(GRAY).set_fill(GRAY, opacity=0.3),
            crispr_cas9.animate.set_color(GRAY).set_opacity(0.3),
            run_time=1.5
        )
        self.wait(2)

class Scene25_PublicOpinionSurvey(Scene):
    """Public opinion survey on gene therapy"""
    def construct(self):
        # Title
        title = Tex("\\textbf{Public Opinion Survey}", font_size=48, color=YELLOW)
        title.move_to(UP * 3.2)

        # Create 100 people icons in 10x10 grid
        people_icons = VGroup()
        rows, cols = 10, 10
        spacing = 0.6

        for row in range(rows):
            for col in range(cols):
                person = Circle(radius=0.08, color=WHITE, fill_opacity=1, fill_color=WHITE)
                person[0].move_to(UP * 0.1)

                x_pos = (col - cols / 2) * spacing + spacing / 2
                y_pos = (rows / 2 - row) * spacing - spacing / 2
                person.move_to(RIGHT * x_pos + UP * y_pos + DOWN * 0.5)

                people_icons.add(person)

        self.play(
            Write(title),
            LaggedStart(*[FadeIn(person) for person in people_icons], lag_ratio=0.01),
            run_time=1
        )
        self.wait(1)

        # Group them randomly (predefined random)
        oppose_indices = (
            2,  5,  8,  13, 14, 19, 20, 23, 27, 28,
            30, 42, 45, 47, 50, 51, 52, 60, 63, 68,
            69, 94, 97, 99, 92
        )
        oppose_people = [people_icons[i] for i in oppose_indices]
        support_people = [people_icons[i] for i in range(100) if i not in oppose_indices]

        # Highlight 75% in green (supporting gene therapy)
        support_label = Tex("\\textbf{75\\% believe in\\\\gene therapy}", font_size=48, color=GREEN)
        support_label.move_to(LEFT * 5)

        self.play(
            *[support_people[i].animate.set_color(GREEN) for i in range(75)],
            Write(support_label),
            run_time=1
```

```
        )
        self.wait(0.5)

        # Show the remaining 25% in red
        oppose_label = Tex("\\textbf{25\\% uncertain\\\\or opposed}", font_size=48, color=RED)
        oppose_label.move_to(RIGHT * 5)

        self.play(
            *[oppose_people[i].animate.set_color(RED) for i in range(25)],
            Write(oppose_label),
            run_time=1
        )

        self.wait(3)

class Scene26_EthicalConcerns(Scene):
    """Ethical concerns about gene therapy"""
    def construct(self):
        # Title
        title = Tex("{\\bfseries Ethical Concerns}", font_size=56, color=YELLOW)
        title.move_to(UP * 3)

        # Concern statements
        concern1_title = Tex("$\\bullet$ \\textbf{``Playing God''}", font_size=36, color=RED)
        concern1_detail = Tex("Changing nature may cause unexpected dangerous results", font_size=28)
        concern1 = VGroup(concern1_title, concern1_detail).arrange(DOWN, buff=0.2, aligned_edge=LEFT)

        concern2_title = Tex("$\\bullet$ \\textbf{Fairness and Access}", font_size=36, color=ORANGE)
        concern2_detail = Tex("Treatments cost millions, risking a permanent genetic divide",
        ↪  font_size=28)
        concern2 = VGroup(concern2_title, concern2_detail).arrange(DOWN, buff=0.2, aligned_edge=LEFT)

        concern3_title = Tex("$\\bullet$ \\textbf{Threat to Human Identity}", font_size=36,
        ↪  color=PURPLE)
        concern3_detail = Tex("Brain-editing errors could cause catastrophic damage", font_size=28)
        concern3 = VGroup(concern3_title, concern3_detail).arrange(DOWN, buff=0.2, aligned_edge=LEFT)

        concerns = VGroup(concern1, concern2, concern3).arrange(DOWN, buff=0.7, aligned_edge=LEFT)
        concerns.move_to(DOWN * 0.3)

        # Animate title
        self.play(Write(title), run_time=1)
        self.wait(0.5)

        # Animate concerns appearing one by one
        self.play(FadeIn(concern1, shift=UP * 0.3), run_time=1)
        self.wait(0.5)
        self.play(FadeIn(concern2, shift=UP * 0.3), run_time=1)
        self.wait(0.5)
        self.play(FadeIn(concern3, shift=UP * 0.3), run_time=1)

        self.wait(3)

```