

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- α 2 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-NleJqY>) – 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.

```
1 from manim import *
2
3 # INTRODUCTION SCENES
4
5 class Scene1_TitleText(Scene):
6     """Opening shot: Somatic Gene Therapy"""
7     def construct(self):
8         # Text animation
9         text = Tex("Science Fiction", font_size=48, color=GREEN)
10        self.play(Write(text))
11        self.wait(2)
12
13        newText = Tex("Somatic Gene Therapy", font_size=48, color=RED)
14        self.play(Transform(text, newText))
15        self.wait(3)
16
17        self.play(FadeOut(text))
18
19 class Scene2_FixingGeneErrorsInACell(Scene):
20     """Scene: Fixing Gene Errors in a Cell"""
21     def construct(self):
22         # Create cell (outer circle) with background
23        cell = Circle(radius=2.5, color=BLUE, stroke_width=3, fill_color=BLUE, fill_opacity=0.3)
24
25        # Create nucleus (inner circle) - smaller size with background
26        nucleus = Circle(radius=0.9, color=PURPLE, stroke_width=3, fill_color=PURPLE,
27            ↪ fill_opacity=0.35)
28
29        # Create problematic DNA (red double helix)
30        # Simple representation using two intertwined curves
31        dna_strand1 = ParametricFunction(
32            lambda t: np.array([
```

```

32         0.3 * np.cos(2 * PI * t),
33         0.8 * t - 0.4,
34         0
35     ]),
36     t_range=[0, 1],
37     color=RED
38 )
39 dna_strand2 = ParametricFunction(
40     lambda t: np.array([
41         -0.3 * np.cos(2 * PI * t),
42         0.8 * t - 0.4,
43         0
44     ]),
45     t_range=[0, 1],
46     color=RED
47 )
48 problematic_dna = VGroup(dna_strand1, dna_strand2)
49
50 # Create text labels
51 cell_label = Tex("$\\textbf{Cell}$", font_size=28, color=BLUE).next_to(cell, DOWN, buff=0.3)
52 problematic_dna_label = Tex("\\textbf{Problematic \\\DNA}", font_size=24,
53     ↪ color=RED).next_to(problematic_dna, RIGHT, buff=0.7)
54
55 # Display cell, nucleus, problematic DNA, and labels at the start
56 self.add(cell, nucleus, problematic_dna, cell_label, problematic_dna_label)
57 self.wait(2)
58
59 # Create fixed DNA (green double helix)
60 fixed_strand1 = ParametricFunction(
61     lambda t: np.array([
62         0.3 * np.cos(2 * PI * t),
63         0.8 * t - 0.4,
64         0
65     ]),
66     t_range=[0, 1],
67     color=GREEN
68 )
69 fixed_strand2 = ParametricFunction(
70     lambda t: np.array([
71         -0.3 * np.cos(2 * PI * t),
72         0.8 * t - 0.4,
73         0
74     ]),
75     t_range=[0, 1],
76     color=GREEN
77 )
78 fixed_dna = VGroup(fixed_strand1, fixed_strand2)
79
80 # Create fixed DNA label
81 fixed_dna_label = Tex("\\textbf{Fixed \\\DNA}", font_size=24, color=GREEN).next_to(fixed_dna,
82     ↪ RIGHT, buff=1.0)
83
84 # Flash effect
85 flash = Flash(problematic_dna.get_center(), color=YELLOW, flash_radius=0.8)
86
87 # Morph DNA to green with pop effect and morph the label
88 self.play(
89     Transform(problematic_dna, fixed_dna),
90     Transform(problematic_dna_label, fixed_dna_label),
91     flash,
92     problematic_dna.animate.scale(1.3).set_color(GREEN)
93 )
94 self.play(problematic_dna.animate.scale(1/1.3))
95 self.wait(2)

```

```

94
95 class Scene3_NoPassing(Scene):
96     """Scene: No Passing"""
97     def construct(self):
98         # Start with the ending status of Scene 2
99         cell = Circle(radius=2.5, color=BLUE, stroke_width=3, fill_color=BLUE, fill_opacity=0.3)
100         nucleus = Circle(radius=0.9, color=PURPLE, stroke_width=3, fill_color=PURPLE,
101             ↪ fill_opacity=0.35)
102
103         # Fixed DNA (green double helix)
104         dna_strand1 = ParametricFunction(
105             lambda t: np.array([
106                 0.3 * np.cos(2 * PI * t),
107                 0.8 * t - 0.4,
108                 0
109             ]),
110             t_range=[0, 1],
111             color=GREEN
112         )
113         dna_strand2 = ParametricFunction(
114             lambda t: np.array([
115                 -0.3 * np.cos(2 * PI * t),
116                 0.8 * t - 0.4,
117                 0
118             ]),
119             t_range=[0, 1],
120             color=GREEN
121         )
122         fixed_dna = VGroup(dna_strand1, dna_strand2)
123
124         # Create labels
125         cell_label = Tex("$\\textbf{Cell}$", font_size=28, color=BLUE).next_to(cell, DOWN, buff=0.3)
126         fixed_dna_label = Tex("$\\textbf{Fixed \\DNA}$", font_size=24, color=GREEN).next_to(fixed_dna,
127             ↪ RIGHT, buff=1.0)
128
129         # Group the cell components
130         cell_group = VGroup(cell, nucleus, fixed_dna)
131
132         # Add everything at the start
133         self.add(cell_group, cell_label, fixed_dna_label)
134
135         # Fade out the text labels
136         self.play(FadeOut(cell_label), FadeOut(fixed_dna_label), run_time=0.25)
137
138         # Create stick figures
139         # Parent (taller, on the left)
140         parent_head = Circle(radius=0.3, color=WHITE, stroke_width=2)
141         parent_body = Line(start=ORIGIN, end=DOWN * 1.5, color=WHITE, stroke_width=2)
142         parent_arms = Line(start=LEFT * 0.5, end=RIGHT * 0.5, color=WHITE, stroke_width=2)
143         parent_legs = VGroup(
144             Line(start=ORIGIN, end=DOWN * 0.8 + LEFT * 0.4, color=WHITE, stroke_width=2),
145             Line(start=ORIGIN, end=DOWN * 0.8 + RIGHT * 0.4, color=WHITE, stroke_width=2)
146         )
147
148         parent = VGroup(parent_head, parent_body, parent_arms, parent_legs)
149         parent_head.next_to(parent_body.get_start(), UP, buff=0.1)
150         parent_arms.next_to(parent_body.get_start(), DOWN, buff=0.3)
151         parent_legs.next_to(parent_body.get_end(), DOWN, buff=0)
152
153         parent.move_to(LEFT * 3 + DOWN * 0.5)
154         parent_label = Tex("$\\textbf{Parent}$", font_size=24, color=WHITE).next_to(parent, DOWN,
155             ↪ buff=0.3)
156
157         # Child (shorter, on the right)

```

```

155 child_head = Circle(radius=0.25, color=WHITE, stroke_width=2)
156 child_body = Line(start=ORIGIN, end=DOWN * 1.0, color=WHITE, stroke_width=2)
157 child_arms = Line(start=LEFT * 0.4, end=RIGHT * 0.4, color=WHITE, stroke_width=2)
158 child_legs = VGroup(
159     Line(start=ORIGIN, end=DOWN * 0.6 + LEFT * 0.3, color=WHITE, stroke_width=2),
160     Line(start=ORIGIN, end=DOWN * 0.6 + RIGHT * 0.3, color=WHITE, stroke_width=2)
161 )
162
163 child = VGroup(child_head, child_body, child_arms, child_legs)
164 child_head.next_to(child_body.get_start(), UP, buff=0.1)
165 child_arms.next_to(child_body.get_start(), DOWN, buff=0.2)
166 child_legs.next_to(child_body.get_end(), DOWN, buff=0)
167
168 child.move_to(RIGHT * 3 + DOWN * 0.3)
169 child_label = Tex("\\textbf{Child}", font_size=24, color=WHITE).next_to(child, DOWN, buff=0.3)
170
171 # Fade in stick figures and their labels
172 self.play(
173     FadeIn(parent), FadeIn(parent_label),
174     FadeIn(child), FadeIn(child_label),
175     cell_group.animate.scale(0.3).shift(parent.get_center() + RIGHT * 1.2),
176     run_time=1
177 )
178
179 # Move cell next to parent
180 self.play(cell_group.animate.move_to(parent.get_center() + RIGHT * 1.2), run_time=0.5)
181 self.wait(0.25)
182
183 # Draw arrow from left to right
184 arrow = Arrow(
185     start=LEFT * 1.5,
186     end=RIGHT * 1.5,
187     color=YELLOW,
188     buff=0,
189     stroke_width=6
190 ).move_to(DOWN * 0.5)
191 self.play(Create(arrow), run_time=0.25)
192
193 # Cell attempts to move along arrow but is stopped by red cross
194 # Create red cross in the middle
195 cross_line1 = Line(start=UP * 0.3 + LEFT * 0.3, end=DOWN * 0.3 + RIGHT * 0.3, color=RED,
196 ↪ stroke_width=8)
197 cross_line2 = Line(start=UP * 0.3 + RIGHT * 0.3, end=DOWN * 0.3 + LEFT * 0.3, color=RED,
198 ↪ stroke_width=8)
199 red_cross = VGroup(cross_line1, cross_line2).move_to(DOWN * 0.5)
200
201 # Animate cell moving and being stopped
202 self.play(
203     cell_group.animate.move_to(cell_group.get_center() + RIGHT * 1.5),
204     run_time=1,
205     rate_func=rush_into
206 )
207
208 # Show red cross appearing and cell bouncing back slightly
209 self.play(Create(red_cross), cell_group.animate.shift(LEFT * 0.3), run_time=0.5)
210 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).

```

1 from manim import *
2
3 # APPLICATION - EYES SCENES
4
5 class Scene4_Chapter1Title(Scene):
6     """Chapter Title: Application - Eyes"""
7     def construct(self):
8         # Text animation
9         chapter = Tex("\\scshape\\bfseries CHAPTER I", font_size=50, color=YELLOW)
10        title = Tex("\\bfseries Genetic Blindness", font_size=64, color=YELLOW)
11        separator = Line(start=LEFT * 3, end=RIGHT * 3, color=ORANGE, stroke_width=4)
12
13        chapter.move_to(UP * 0.75)
14        title.move_to(DOWN * 0.75)
15        separator.move_to(ORIGIN)
16
17        self.play(
18            Write(chapter, run_time=1.5),
19            Create(separator, run_time=1.5),
20            Write(title, run_time=1.5)
21        )
22
23        self.wait(3)
24
25 class Scene5_LuxturnaAndLCATitle(Scene):
26     """Title: Luxturna and LCA"""
27     def construct(self):
28         # Text animation
29         title = Tex("\\bfseries Luxturna", font_size=64, color=BLUE)
30         subtitle = Tex("A gene therapy for", font_size=36)
31         disease = Tex("\\bfseries Leber Congenital Amaurosis (LCA)", font_size=40, color=RED)
32
33         title.move_to(UP * 1)
34         subtitle.move_to(DOWN * 0.2)
35         disease.move_to(DOWN * 0.8)
36
37         self.play(Write(title, run_time=0.75))
38         self.wait(3)
39         self.play(Write(subtitle), Write(disease), run_time=0.75)
40
41         self.wait(3)
42
43 class Scene6_LuxturnaPacking(Scene):
44     """Luxturna Packing Animation"""
45     def construct(self):
46         scene_title = Tex("\\bfseries Luxturna Mechanism", font_size=48, color=YELLOW)
47         scene_title.move_to(UP * 3)
48         self.play(Write(scene_title), run_time=1)
49         self.wait(0.5)
50
51         # 1. Create cDNA (green single strand) on the left
52         cdna = ParametricFunction(
53             lambda t: np.array([
54                 0.2 * np.sin(4 * PI * t),
55                 2 * t - 1,
56                 0
57             ]),
58             t_range=[0, 1],
59             color=GREEN,
60             stroke_width=6
61         )
62

```

```

63 cdna_label1 = Tex("\\textbf{cDNA}", font_size=28, color=GREEN)
64 cdna_label2 = Tex("\\textbf{(functional RPE65)}", font_size=24, color=GREEN)
65 cdna_labels = VGroup(cdna_label1, cdna_label2).arrange(DOWN, buff=0.2)
66
67 cdna.move_to(LEFT * 4)
68 cdna_labels.next_to(cdna, DOWN, buff=0.5)
69
70 # 2. Create AAV viral vector (purple capsid) on the right
71 # Represent as a hexagonal/circular capsid with inner space
72 aav_capsid = RegularPolygon(n=6, radius=1.2, color=PURPLE, stroke_width=4, fill_color=PURPLE,
73 ↪ fill_opacity=0.2)
74
75 # Add spike proteins around the capsid
76 spike_positions = [
77     UP * 1.2,
78     UP * 0.6 + RIGHT * 1.0,
79     DOWN * 0.6 + RIGHT * 1.0,
80     DOWN * 1.2,
81     DOWN * 0.6 + LEFT * 1.0,
82     UP * 0.6 + LEFT * 1.0
83 ]
84
85 spikes = VGroup()
86 for pos in spike_positions:
87     spike = Triangle(color=PURPLE, fill_color=PURPLE, fill_opacity=0.8)
88     spike.scale(0.15)
89     spike.move_to(pos)
90     # Point spike outward from center
91     angle = np.arctan2(pos[1], pos[0])
92     spike.rotate(angle + PI/2)
93     spikes.add(spike)
94
95 aav_vector = VGroup(aav_capsid, spikes)
96
97 aav_label = Tex("\\textbf{AAV therapeutic\\\\vector}", font_size=28, color=PURPLE)
98
99 aav_vector.move_to(RIGHT * 4)
100 aav_label.next_to(aav_vector, DOWN, buff=0.5)
101
102 # Display both components
103 self.play(
104     Create(cdna),
105     Write(cdna_labels),
106     run_time=0.7
107 )
108 self.wait(1.5)
109 self.play(
110     Create(aav_vector),
111     Write(aav_label),
112     run_time=0.7
113 )
114 self.wait(2)
115
116 # 3. Move cDNA into the viral vector
117 # Scale down cDNA to fit inside
118 self.play(
119     cdna.animate.scale(0.4).move_to(aav_vector.get_center()),
120     FadeOut(cdna_labels),
121     run_time=1
122 )
123 self.wait(0.5)
124
125 # Group the combination
126 aav_with_cdna = VGroup(aav_vector, cdna)

```

```

126
127 # 4. Zoom and move to center, transform label
128 aav_new_label = Tex("\\textbf{AAV2 serotype}", font_size=32, color=PURPLE)
129 aav_new_label.move_to(DOWN * 2.5)
130
131 self.play(
132     aav_with_cdna.animate.scale(1.5).move_to(ORIGIN),
133     Transform(aav_label, aav_new_label),
134     run_time=1
135 )
136 self.wait(3)
137
138 class Scene7_LuxturnaInjection(Scene):
139     """Luxturna Injection Animation"""
140     def construct(self):
141         scene_title = Tex("\\bfseries Luxturna Mechanism", font_size=48, color=YELLOW)
142         scene_title.move_to(UP * 3)
143         self.add(scene_title)
144
145         # 1. Start from where Scene 6 ended - recreate the AAV2 with cDNA at center
146         aav_capsid = RegularPolygon(n=6, radius=1.2, color=PURPLE, stroke_width=4, fill_color=PURPLE,
147             ↪ fill_opacity=0.2)
148
149         spike_positions = [
150             UP * 1.2,
151             UP * 0.6 + RIGHT * 1.0,
152             DOWN * 0.6 + RIGHT * 1.0,
153             DOWN * 1.2,
154             DOWN * 0.6 + LEFT * 1.0,
155             UP * 0.6 + LEFT * 1.0
156         ]
157
158         spikes = VGroup()
159         for pos in spike_positions:
160             spike = Triangle(color=PURPLE, fill_color=PURPLE, fill_opacity=0.8)
161             spike.scale(0.15)
162             spike.move_to(pos)
163             angle = np.arctan2(pos[1], pos[0])
164             spike.rotate(angle + PI/2)
165             spikes.add(spike)
166
167         aav_vector = VGroup(aav_capsid, spikes)
168
169         cdna = ParametricFunction(
170             lambda t: np.array([
171                 0.2 * np.sin(4 * PI * t),
172                 2 * t - 1,
173                 0
174             ]),
175             t_range=[0, 1],
176             color=GREEN,
177             stroke_width=6
178         ).scale(0.4)
179
180         aav_with_cdna = VGroup(aav_vector, cdna).scale(1.5).move_to(ORIGIN)
181         aav_label = Tex("\\textbf{AAV2 serotype}", font_size=32, color=PURPLE).move_to(DOWN * 2.5)
182
183         self.add(aav_with_cdna, aav_label)
184
185         # 2. Zoom out AAV2 and move to left, create impaired cell on right-center
186         self.play(
187             aav_with_cdna.animate.scale(0.5).move_to(LEFT * 4),
188             FadeOut(aav_label),
189             run_time=1

```

```

189 )
190
191 # Create cell with impaired DNA (red) in nucleus
192 cell = Circle(radius=2, color=BLUE, stroke_width=3, fill_color=BLUE, fill_opacity=0.1)
193 nucleus = Circle(radius=0.8, color=PURPLE, stroke_width=3, fill_color=PURPLE,
194 ↪ fill_opacity=0.15)
195
196 # Impaired DNA (red)
197 impaired_dna = ParametricFunction(
198     lambda t: np.array([
199         0.2 * np.sin(4 * PI * t),
200         0.6 * t - 0.3,
201         0
202     ]),
203     t_range=[0, 1],
204     color=RED,
205     stroke_width=4
206 )
207
208 cell_group = VGroup(cell, nucleus, impaired_dna)
209 cell_group.move_to(RIGHT * 1.5)
210
211 # Add cell label for impaired state
212 cell_label = Tex("\\textbf{Cell (impaired DNA)}", font_size=32, color=RED)
213 cell_label.next_to(cell_group, DOWN, buff=0.3)
214
215 self.play(Create(cell_group), Write(cell_label), run_time=1)
216
217 # 3. Vector moves towards cell and releases cDNA
218 self.play(
219     aav_with_cdna.animate.move_to(cell.get_left() + LEFT * 0.5),
220     run_time=1.2
221 )
222
223 # Extract and scale up the cDNA for delivery
224 cdna_delivery = cdna.copy().scale(2)
225
226 # Create fixed cell label
227 fixed_cell_label = Tex("\\textbf{Cell (fixed DNA)}", font_size=32, color=GREEN)
228 fixed_cell_label.next_to(cell_group, DOWN, buff=0.3)
229
230 self.play(
231     cdna_delivery.animate.move_to(nucleus.get_center()),
232     impaired_dna.animate.set_color(GREEN),
233     Transform(cell_label, fixed_cell_label),
234     run_time=1.5
235 )
236
237 # Remove the delivery cDNA and keep the transformed one
238 self.remove(cdna_delivery)
239
240 # 4. Vector fades out, cell moves to center and zooms slightly
241 self.play(FadeOut(aav_with_cdna), run_time=0.5)
242
243 self.play(
244     cell_group.animate.scale(0.9).move_to(ORIGIN),
245     cell_label.animate.next_to(ORIGIN + DOWN * 2, DOWN, buff=0),
246     run_time=1
247 )
248
249 # 5. Cell produces functional retinoid isomerase (blue hexagons)
250 enzyme_label = Tex("\\textbf{Functional Retinoid\\Isomerase (RPE65)}", font_size=32,
251 ↪ color=BLUE)
252 enzyme_label.move_to(DOWN * 3)

```

```

251
252     # Create enzymes with fixed random positions
253     enzyme_positions = [
254         UP * 2 + LEFT * 5,
255         UP * 1.5 + RIGHT * 5.5,
256         UP * 0.5 + LEFT * 6,
257         DOWN * 0.8 + RIGHT * 6,
258         DOWN * 2 + LEFT * 5.5,
259         DOWN * 2.5 + RIGHT * 4.5,
260         UP * 2.8 + LEFT * 2,
261         UP * 2.5 + RIGHT * 3,
262         DOWN * 3 + LEFT * 3,
263         DOWN * 2.8 + RIGHT * 2.5,
264         UP * 1 + LEFT * 3.5,
265         DOWN * 1.5 + LEFT * 4
266     ]
267
268     enzymes = VGroup()
269     animations = []
270
271     for i, pos in enumerate(enzyme_positions):
272         enzyme = RegularPolygon(n=6, radius=0.15, color=BLUE, fill_color=BLUE, fill_opacity=0.7)
273         enzyme.move_to(cell.get_center())
274         enzymes.add(enzyme)
275
276         # Stagger the animations slightly
277         animations.append(
278             AnimationGroup(
279                 enzyme.animate.move_to(pos),
280                 lag_ratio=0.1
281             )
282         )
283
284     self.add(enzymes)
285
286     # Animate enzymes leaving the cell
287     self.play(
288         LaggedStart(*[enzyme.animate.move_to(pos) for enzyme, pos in zip(enzymes,
289                                     ↪ enzyme_positions)],
289                     lag_ratio=0.08),
290         Write(enzyme_label),
291         run_time=2.8
292     )
293
294     self.wait(2)
295
296 class Scene8_ClinicalResultsEyes(Scene):
297     """Clinical Results Text"""
298     def construct(self):
299         # Title
300         title = Tex("\\bfseries Clinical Results", font_size=56, color=YELLOW)
301         title.move_to(UP * 2.5)
302
303         # Result statements
304         result1 = Tex("\\bullet$ Significant improvements in retinal function", font_size=32)
305         result2 = Tex("\\bullet$ Increased light sensitivity", font_size=32)
306         result3 = Tex("\\bullet$ Enhanced pupillary responses", font_size=32)
307         result4 = Tex("\\bullet$ Improved vision in dim environments", font_size=32, color=GREEN)
308
309         results = VGroup(result1, result2, result3, result4).arrange(DOWN, buff=0.5,
310                                     ↪ aligned_edge=LEFT)
311         results.move_to(DOWN * 0.5)
312
313         # Animate title

```

```

313     self.play(Write(title), run_time=1)
314     self.wait(0.5)
315
316     # Animate results appearing one by one
317     self.play(FadeIn(result1, shift=UP * 0.3), run_time=0.8)
318     self.wait(0.3)
319     self.play(FadeIn(result2, shift=UP * 0.3), run_time=0.8)
320     self.wait(0.3)
321     self.play(FadeIn(result3, shift=UP * 0.3), run_time=0.8)
322     self.wait(0.3)
323     self.play(FadeIn(result4, shift=UP * 0.3), run_time=0.8)
324
325     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.

```

1  from manim import *
2
3  # BLOOD DISORDER APPLICATION SCENES
4
5  class Scene9_Chapter2Title(Scene):
6      """Chapter 2 Title: Gene Therapy for Blood Disorders"""
7      def construct(self):
8          # Text animation
9          chapter = Tex("\\scshape\\bfseries CHAPTER II)", font_size=50, color=YELLOW)
10         title = Tex("\\bfseries Blood Disorders)", font_size=64, color=YELLOW)
11         separator = Line(start=LEFT * 3, end=RIGHT * 3, color=ORANGE, stroke_width=4)
12
13         chapter.move_to(UP * 0.75)
14         title.move_to(DOWN * 0.75)
15         separator.move_to(ORIGIN)
16
17         self.play(
18             Write(chapter, run_time=1.5),
19             Create(separator, run_time=1.5),
20             Write(title, run_time=1.5)
21         )
22
23         self.wait(3)
24
25  class Scene10_SickleCellDiseaseTitle(Scene):
26      """Sickle Cell Disease Title Scene"""
27      def construct(self):
28          # Text animation
29          title = Tex("\\bfseries Sickle Cell Disease)", font_size=72, color=YELLOW)
30
31          self.play(Write(title, run_time=1))
32
33          self.wait(3)
34
35  class Scene11_CRISPRTreatment(Scene):
36      """CRISPR-Cas9 Treatment for Sickle Cell Disease"""
37      def construct(self):
38          # Part 1: Show BCL11A gene suppressing fetal hemoglobin
39          bcl11a_gene = Rectangle(width=2, height=0.8, color=RED, fill_color=RED, fill_opacity=0.3)
40          bcl11a_label = Tex("\\textbf{BCL11A}\\gene)", font_size=24, color=RED)
41          bcl11a_label.next_to(bcl11a_gene, UP, buff=0.3)
42
43          bcl11a_group = VGroup(bcl11a_gene, bcl11a_label)

```

```

44     bcl11a_group.move_to(UP * 2 + LEFT * 3)
45
46     # Fetal hemoglobin (suppressed, shown as small gray circles)
47     fetal_hb_suppressed = VGroup(*[
48         Circle(radius=0.15, color=GRAY, fill_color=GRAY, fill_opacity=0.4)
49         for _ in range(3)
50     ]).arrange(RIGHT, buff=0.3)
51     fetal_hb_suppressed.move_to(DOWN * 1 + LEFT * 3)
52
53     fetal_label = Tex("\\textbf{Fetal Hb\\\\(suppressed)}", font_size=20, color=GRAY)
54     fetal_label.next_to(fetal_hb_suppressed, DOWN, buff=0.3)
55
56     # Suppression arrow
57     suppression_arrow = Arrow(
58         start=bcl11a_gene.get_bottom(),
59         end=fetal_hb_suppressed.get_top(),
60         color=RED,
61         stroke_width=6
62     )
63
64     self.play(
65         Create(bcl11a_group),
66         Create(suppression_arrow),
67         Create(fetal_hb_suppressed),
68         Write(fetal_label),
69         run_time=1.5
70     )
71     self.wait(1)
72
73     # Part 2: Show CRISPR-Cas9 coming in to edit BCL11A
74     crispr_cas9 = VGroup(
75         Rectangle(width=1.5, height=0.6, color=BLUE, fill_color=BLUE, fill_opacity=0.5),
76         Tex("\\textbf{CRISPR-\\\\Cas9}", font_size=20, color=BLUE)
77     )
78     crispr_cas9[1].move_to(crispr_cas9[0].get_center())
79     crispr_cas9.move_to(UP * 2 + RIGHT * 2)
80
81     self.play(Create(crispr_cas9), run_time=1)
82     self.wait(0.5)
83
84     # CRISPR moves to BCL11A
85     self.play(crispr_cas9.animate.move_to(bcl11a_gene.get_center()), run_time=1.5)
86
87     # Flash effect showing gene editing
88     flash = Flash(bcl11a_gene.get_center(), color=YELLOW, flash_radius=1)
89     self.play(flash)
90
91     # BCL11A becomes deactivated (crossed out)
92     cross_line1 = Line(
93         start=bcl11a_gene.get_corner(UL),
94         end=bcl11a_gene.get_corner(DR),
95         color=YELLOW,
96         stroke_width=6
97     )
98     cross_line2 = Line(
99         start=bcl11a_gene.get_corner(UR),
100        end=bcl11a_gene.get_corner(DL),
101        color=YELLOW,
102        stroke_width=6
103    )
104
105    deactivated_label = Tex("\\textbf{BCL11A\\\\(deactivated)}", font_size=24, color=GRAY)
106    deactivated_label.move_to(bcl11a_label.get_center())

```

```

107
108     self.play(
109         Create(cross_line1),
110         Create(cross_line2),
111         Transform(bcl11a_label, deactivated_label),
112         FadeOut(crispr_cas9),
113         suppression_arrow.animate.set_color(GRAY).set_opacity(0.3),
114         run_time=1.5
115     )
116     self.wait(1)
117
118     # Part 3: Fetal hemoglobin production resumes
119     fetal_hb_active = VGroup(*[
120         Circle(radius=0.2, color=GREEN, fill_color=GREEN, fill_opacity=0.7)
121         for _ in range(6)
122     ]).arrange_in_grid(rows=2, cols=3, buff=0.3)
123     fetal_hb_active.move_to(fetal_hb_suppressed.get_center() + DOWN * 0.5)
124
125     active_label = Tex("\\textbf{Fetal Hb\\\\(high-level)}", font_size=20, color=GREEN)
126     active_label.next_to(fetal_hb_active, DOWN, buff=0.3)
127
128     self.play(
129         Transform(fetal_hb_suppressed, fetal_hb_active),
130         Transform(fetal_label, active_label),
131         run_time=1.5
132     )
133     self.wait(1)
134
135     # Part 4: Show effect on red blood cells
136     # Fade everything out
137     self.play(
138         FadeOut(bcl11a_group),
139         FadeOut(cross_line1),
140         FadeOut(cross_line2),
141         FadeOut(suppression_arrow),
142         FadeOut(fetal_hb_suppressed),
143         FadeOut(fetal_label),
144         run_time=1
145     )
146
147     # 1. Create sickle cells and normal hemoglobin scattered across screen
148     # Sickle cells (red crescents)
149     sickle_positions = [
150         UP * 2.5 + LEFT * 4,
151         UP * 1.5 + LEFT * 1,
152         UP * 0.5 + LEFT * 5,
153         DOWN * 0.5 + LEFT * 2.5,
154         DOWN * 1.5 + LEFT * 5.5,
155         UP * 2 + RIGHT * 1,
156         DOWN * 2 + LEFT * 0.5,
157     ]
158
159     sickle_cells = VGroup()
160     for pos in sickle_positions:
161         arc1 = Arc(radius=0.3, angle=PI, color=RED, stroke_width=3, fill_color=RED,
162             ↪ fill_opacity=0.4)
163         arc2 = Arc(radius=0.25, angle=-PI, color=RED, stroke_width=3, fill_color=RED,
164             ↪ fill_opacity=0.4).next_to(arc1, RIGHT, buff=0)
165         sickle = VGroup(arc1, arc2).rotate(PI/4)
166         sickle.move_to(pos)
167         sickle_cells.add(sickle)
168
169     # Normal hemoglobin (green circles)
170     normal_hb_positions = [

```

```

169         UP * 3 + LEFT * 2,
170         UP * 1 + RIGHT * 4,
171         DOWN * 1 + RIGHT * 5,
172         DOWN * 2.5 + RIGHT * 2,
173         UP * 0.5 + RIGHT * 2,
174     ]
175
176     normal_hbs = VGroup()
177     for pos in normal_hb_positions:
178         normal_hb = Circle(radius=0.25, color=GREEN, fill_color=GREEN, fill_opacity=0.5,
179                             ↪ stroke_width=3)
180         normal_hb.move_to(pos)
181         normal_hbs.add(normal_hb)
182
183     # Legend at bottom left
184     legend_box = Rectangle(width=3.5, height=1.8, color=WHITE, stroke_width=2)
185     legend_box.move_to(DOWN * 2.7 + LEFT * 4.5)
186
187     # Legend items
188     sickle_legend_icon = VGroup(
189         Arc(radius=0.15, angle=PI, color=RED, stroke_width=2, fill_color=RED, fill_opacity=0.4),
190     )
191     sickle_legend_icon[0].next_to(sickle_legend_icon[0], RIGHT, buff=0)
192     sickle_legend_label = Tex("Sickle Hb", font_size=18, color=RED)
193     sickle_legend = VGroup(sickle_legend_icon, sickle_legend_label).arrange(RIGHT, buff=0.3)
194
195     normal_legend_icon = Circle(radius=0.15, color=GREEN, fill_color=GREEN, fill_opacity=0.5,
196                                 ↪ stroke_width=2)
197     normal_legend_label = Tex("Normal Hb", font_size=18, color=GREEN)
198     normal_legend = VGroup(normal_legend_icon, normal_legend_label).arrange(RIGHT, buff=0.3)
199
200     fetal_legend_icon = Circle(radius=0.15, color=BLUE, fill_color=BLUE, fill_opacity=0.5,
201                                ↪ stroke_width=2)
202     fetal_legend_label = Tex("Fetal Hb", font_size=18, color=BLUE)
203     fetal_legend = VGroup(fetal_legend_icon, fetal_legend_label).arrange(RIGHT, buff=0.3)
204
205     legend_items = VGroup(sickle_legend, normal_legend, fetal_legend).arrange(DOWN,
206                                     ↪ aligned_edge=LEFT, buff=0.2)
207     legend_items.move_to(legend_box.get_center())
208
209     legend_group = VGroup(legend_box, legend_items)
210
211     # 2. Counter at bottom right showing functional hemoglobin percentage
212     counter_box = Rectangle(width=3, height=1.2, color=WHITE, stroke_width=2)
213     counter_box.move_to(DOWN * 2.9 + RIGHT * 4.5)
214
215     counter_label = Tex("\\textbf{Functional Hb}", font_size=20)
216     counter_label.move_to(counter_box.get_center() + UP * 0.3)
217
218     initial_percentage = int((len(normal_hbs) / (len(sickle_cells) + len(normal_hbs))) * 100)
219     counter_value = Tex(f"\\textbf{{{initial_percentage}}\\%}", font_size=32, color=YELLOW)
220     counter_value.move_to(counter_box.get_center() + DOWN * 0.25)
221
222     counter_group = VGroup(counter_box, counter_label, counter_value)
223
224     # Fade in everything
225     self.play(
226         Create(sickle_cells),
227         Create(normal_hbs),
228         Create(legend_group),
229         Create(counter_group),
230         run_time=1.5
231     )
232     self.wait(1)

```

```

229
230 # 3. Fade in fetal hemoglobin (blue circles) and update counter
231 fetal_hb_positions = [
232     UP * 2.8 + RIGHT * 3.5,
233     UP * 0.8 + LEFT * 3.5,
234     UP * 1.8 + RIGHT * 5.5,
235     DOWN * 0.3 + RIGHT * 0.5,
236     DOWN * 1.8 + RIGHT * 4.5,
237     DOWN * 2.5 + LEFT * 3,
238     UP * 3.2 + LEFT * 0.5,
239     DOWN * 0.8 + LEFT * 4.5,
240 ]
241
242 fetal_hbs = VGroup()
243 for pos in fetal_hb_positions:
244     fetal_hb = Circle(radius=0.25, color=BLUE, fill_color=BLUE, fill_opacity=0.5,
245                       ↪ stroke_width=3)
246     fetal_hb.move_to(pos)
247     fetal_hbs.add(fetal_hb)
248
249 # Calculate new percentage
250 new_percentage = int((len(normal_hbs) + len(fetal_hbs)) / (len(sickle_cells) + len(normal_hbs)
251 ↪ + len(fetal_hbs)) * 100)
252 new_counter_value = Tex(f"\\textbf{{{new_percentage}}\\%}", font_size=32, color=GREEN)
253 new_counter_value.move_to(counter_value.get_center())
254
255 self.play(
256     FadeIn(fetal_hbs, lag_ratio=0.1),
257     Transform(counter_value, new_counter_value),
258     run_time=2
259 )
260 self.wait(1.5)
261
262 # Final message
263 cure_text = Tex("\\textbf{Malfunctioned Haemoglobin Diluted}", font_size=40, color=WHITE)
264 cure_text_box = Rectangle(width=8, height=1, color=GREEN, stroke_width=3, fill_color=GREEN,
265 ↪ fill_opacity=0.3)
266 cure_text.move_to(ORIGIN)
267
268 self.play(Create(cure_text_box), Write(cure_text), run_time=1)
269 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.

```

1 from manim import *
2 import random
3
4 # BLOOD CANCER APPLICATION SCENES
5
6 class Scene12_Chapter3Title(Scene):
7     """Chapter 3 Title: Leukaemia and Lymphoma"""
8     def construct(self):
9         # Text animation
10         chapter = Tex("\\scshape\\bfseries CHAPTER III)", font_size=50, color=YELLOW)
11         title = Tex("\\bfseries Leukaemia and Lymphoma)", font_size=64, color=YELLOW)
12         separator = Line(start=LEFT * 3, end=RIGHT * 3, color=ORANGE, stroke_width=4)
13
14         chapter.move_to(UP * 0.75)
15         title.move_to(DOWN * 0.75)

```

```

16     separator.move_to(ORIGIN)
17
18     self.wait(1) # for editing
19
20     self.play(
21         Write(chapter, run_time=1.5),
22         Create(separator, run_time=1.5),
23         Write(title, run_time=1.5)
24     )
25
26     self.wait(3)
27
28 class Scene13_WBCDivision(Scene):
29     """White Blood Cell Division and Mutation Introduction Scene"""
30     def construct(self):
31         # Create initial white blood cell
32         initial_cell = Circle(radius=0.3, color=WHITE, fill_opacity=0.8, fill_color=WHITE,
33                               stroke_width=2)
34         initial_cell.move_to(ORIGIN)
35
36         # Store all cells in a list
37         cells = [initial_cell]
38
39         self.add(initial_cell)
40         self.wait(1)
41
42         # Division loop
43         max_cells = 100
44         division_interval = 0.15 # Time between divisions
45         cell_radius = 0.3
46
47     def find_non_overlapping_position(parent_pos, existing_cells, max_attempts=50):
48         """Find a position for new cell that doesn't overlap with existing cells"""
49         for attempt in range(max_attempts):
50             angle = np.random.uniform(0, 2 * np.pi)
51             distance = cell_radius * 2.5 # Start at safe distance
52             offset = np.array([np.cos(angle), np.sin(angle), 0]) * distance
53             new_pos = parent_pos + offset
54
55             # Check if position is valid (not overlapping and within frame)
56             if abs(new_pos[0]) < config.frame_width / 2 - cell_radius and \
57                abs(new_pos[1]) < config.frame_height / 2 - cell_radius:
58
59                 # Check overlap with existing cells
60                 overlapping = False
61                 for cell in existing_cells:
62                     dist = np.linalg.norm(new_pos - cell.get_center())
63                     if dist < cell_radius * 2.2: # Minimum safe distance
64                         overlapping = True
65                         break
66
67                 if not overlapping:
68                     return new_pos
69
70             # If no good position found, return random position with larger offset
71             angle = np.random.uniform(0, 2 * np.pi)
72             distance = cell_radius * 4
73             offset = np.array([np.cos(angle), np.sin(angle), 0]) * distance
74             return parent_pos + offset
75
76     while len(cells) < max_cells:
77         # Select a random cell to divide from all existing cells
78         if len(cells) > 0:
79             parent = random.choice(cells)

```

```

79         parent_pos = parent.get_center()
80
81         # Find positions for two daughter cells
82         pos1 = find_non_overlapping_position(parent_pos, cells)
83
84         # Create first daughter cell
85         new_cell1 = Circle(radius=cell_radius, color=WHITE, fill_opacity=0.8,
86                             ↪ fill_color=WHITE, stroke_width=2)
87         new_cell1.move_to(parent_pos)
88
89         # Temporarily add to list to check for second position
90         temp_cells = cells + [new_cell1]
91         pos2 = find_non_overlapping_position(parent_pos, temp_cells)
92
93         # Create second daughter cell
94         new_cell2 = Circle(radius=cell_radius, color=WHITE, fill_opacity=0.8,
95                             ↪ fill_color=WHITE, stroke_width=2)
96         new_cell2.move_to(parent_pos)
97
98         # Remove parent cell and add daughter cells
99         self.remove(parent)
100        cells.remove(parent)
101
102        cells.append(new_cell1)
103        cells.append(new_cell2)
104
105        self.add(new_cell1, new_cell2)
106
107        # Animate cells moving to their positions
108        self.play(
109            new_cell1.animate.move_to(pos1),
110            new_cell2.animate.move_to(pos2),
111            run_time=0.3,
112            rate_func=smooth
113        )
114
115        if len(cells) >= max_cells:
116            break
117
118        self.wait(2)
119
120    class Scene14_CARTCellTherapy(Scene):
121        """CAR T-cell Therapy Process"""
122        def construct(self):
123            # Text animation
124            title = Tex("\\bfseries CAR-T Cell Therapy", font_size=72, color=YELLOW)
125
126            self.wait(2) # for editing
127
128            self.play(Write(title, run_time=1))
129
130            self.wait(3)
131
132    class Scene15_HighRateBut(Scene):
133        def construct(self):
134            # Text animation
135            text = Tex("\\textbf{High Remission Rate}", font_size=64, color=YELLOW)
136            text.move_to(UP * 0.5)
137            self.wait(1.5)
138            self.play(Write(text, run_time=1))
139            self.wait(1)
140
141            but_text = Tex("\\textbf{But...?}", font_size=64, color=RED)
142            but_text.move_to(DOWN * 0.5)

```

```

141         self.play(Write(but_text, run_time=1))
142         self.wait(2)
143
144     class Scene16_CytokineStorm(Scene):
145         """Cytokine Storm Explanation Scene"""
146         def construct(self):
147             # 1. T cell
148             t_cell = Circle(radius=0.5, color=BLUE, fill_color=BLUE, fill_opacity=0.5, stroke_width=3)
149             t_cell.move_to(UP * 2.5)
150             t_cell_label = Tex("\\textbf{T Cell}", font_size=24, color=BLUE).next_to(t_cell, DOWN,
151             ↪ buff=0.3)
152             t_cell_group = VGroup(t_cell, t_cell_label)
153
154             # 2. Cancer Cell
155             cancer_cell = Circle(radius=0.5, color=RED, fill_color=RED, fill_opacity=0.5, stroke_width=3)
156             cancer_cell.move_to(DOWN * 2.5 + LEFT * 3)
157             cancer_cell_label = Tex("\\textbf{Cancer Cell}", font_size=24, color=RED).next_to(cancer_cell,
158             ↪ DOWN, buff=0.3)
159             cancer_cell_group = VGroup(cancer_cell, cancer_cell_label)
160
161             # 3. Healthy Cell
162             healthy_cell = Circle(radius=0.5, color=GREEN, fill_color=GREEN, fill_opacity=0.5,
163             ↪ stroke_width=3)
164             healthy_cell.move_to(DOWN * 2.5 + RIGHT * 3)
165             healthy_cell_label = Tex("\\textbf{Healthy Cell}", font_size=24,
166             ↪ color=GREEN).next_to(healthy_cell, DOWN, buff=0.3)
167             healthy_cell_group = VGroup(healthy_cell, healthy_cell_label)
168
169             # arrows
170             arrow_to_cancer = Arrow(
171                 start=t_cell_group.get_bottom(),
172                 end=cancer_cell_group.get_top() + DOWN * 0.1,
173                 color=YELLOW,
174                 buff=0.5,
175                 stroke_width=6
176             )
177
178             arrow_to_healthy = Arrow(
179                 start=t_cell_group.get_bottom(),
180                 end=healthy_cell_group.get_top() + DOWN * 0.1,
181                 color=YELLOW,
182                 buff=0.5,
183                 stroke_width=6
184             )
185
186             # text
187             text = Tex("\\textbf{Cytokine Storm}", font_size=48, color=ORANGE)
188             text.next_to(arrow_to_healthy.get_center(), RIGHT, buff=1)
189
190             self.wait(1)
191             self.play(
192                 Create(t_cell), Write(t_cell_label),
193                 Create(cancer_cell), Write(cancer_cell_label),
194                 Create(healthy_cell), Write(healthy_cell_label),
195                 run_time=0.5
196             )
197             self.wait(1)
198             self.play(Create(arrow_to_cancer), run_time=0.5)
199             self.play(Create(arrow_to_healthy), Write(text), run_time=0.5)
200
201             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).

```
1 from manim import *
2
3 # NEUROMUSCULAR DISEASE APPLICATION SCENES
4
5 class Scene17_Chapter4Title(Scene):
6     def construct(self):
7         # Text animation
8         chapter = Tex("\scshape\\bfseries CHAPTER IV", font_size=50, color=YELLOW)
9         title = Tex("\bfseries Spinal Muscular Atrophy", font_size=64, color=YELLOW)
10        separator = Line(start=LEFT * 3, end=RIGHT * 3, color=ORANGE, stroke_width=4)
11
12        chapter.move_to(UP * 0.75)
13        title.move_to(DOWN * 0.75)
14        separator.move_to(ORIGIN)
15
16        self.play(
17            Write(chapter, run_time=1.5),
18            Create(separator, run_time=1.5),
19            Write(title, run_time=1.5)
20        )
21
22        self.wait(3)
23
24 class Scene18_BackupSMN2Gene(Scene):
25     def construct(self):
26         # title
27         title = Tex("\textbf{SMN2 Gene As a Backup}", font_size=48, color=YELLOW)
28         title.move_to(UP * 3)
29         self.play(Write(title), run_time=1)
30         self.wait(1)
31
32         # A segment of SMN2 RNA, represented by two parallel wavy lines (sine waves)
33         x_start = -6
34         x_end = 6
35         amplitude = 0.3
36         frequency = 1
37         wave_func = lambda x: amplitude * np.sin(frequency * x)
38         smn2_rna = VGroup(
39             ParametricFunction(
40                 lambda t: np.array([t, wave_func(t) + 0.3, 0]),
41                 t_range=[x_start, x_end],
42                 color=BLUE,
43                 stroke_width=4
44             ),
45             ParametricFunction(
46                 lambda t: np.array([t, wave_func(t) - 0.3, 0]),
47                 t_range=[x_start, x_end],
48                 color=BLUE,
49                 stroke_width=4
50             )
51         )
52         smn2_rna.scale(1.3)
53
54         smn2_label = Tex("\textbf{SMN2 RNA}", font_size=32, color=BLUE).next_to(smn2_rna, UP,
55                                     ↪ buff=0.5)
56         smn2_label.shift(LEFT * 1.7 + DOWN * 1)
57
58         self.add(smn2_rna)
```

```

58 self.play(Write(smn2_label), run_time=1)
59 self.wait(1)
60
61 # An error marker (red exclamation mark, with a circle around) pops on the RNA
62 exclamation_mark = Tex("!", font_size=56, color=RED)
63 error_circle = Circle(radius=0.3, color=RED, stroke_width=3)
64 error_marker = VGroup(error_circle, exclamation_mark).move_to(smn2_rna.get_center() + RIGHT *
↳ 2 + UP * 0.35)
65 error_flash = Flash(error_marker.get_center(), color=YELLOW, flash_radius=0.4,
↳ line_stroke_width=6)
66
67 self.play(Create(error_marker), error_flash, run_time=1)
68 self.wait(1)
69
70 # two dashed lines to indicate splicing
71 splice_line1 = DashedLine(
72     start=smn2_rna[0].point_from_proportion(0.5),
73     end=smn2_rna[1].point_from_proportion(0.5),
74     color=WHITE,
75     stroke_width=4,
76     dash_length=0.06,
77     buff=0
78 )
79 splice_line2 = DashedLine(
80     start=smn2_rna[0].point_from_proportion(0.8),
81     end=smn2_rna[1].point_from_proportion(0.8),
82     color=WHITE,
83     stroke_width=4,
84     dash_length=0.06,
85     buff=0
86 )
87 exon7_text = Tex("\\textbf{Exon 7} \\\\excluded", font_size=28,
↳ color=RED).next_to(splice_line2, RIGHT, buff=0.3)
88 exon7_text.next_to(error_marker, DOWN, buff=0.2)
89 self.play(Create(splice_line1), Create(splice_line2), Write(exon7_text), run_time=1)
90 self.wait(1)
91
92 # arrow down
93 arrow = Arrow(
94     start=splice_line1.get_bottom() + DOWN * 0.2,
95     end=splice_line1.get_bottom() + DOWN * 1.2,
96     color=YELLOW,
97     buff=0,
98     stroke_width=6
99 )
100
101 # 3/4 circle to represent truncated protein
102 truncated_protein = Arc(
103     radius=0.4,
104     start_angle=PI / 4,
105     angle=3 * PI / 2,
106     color=ORANGE,
107     stroke_width=6
108 ).move_to(arrow.get_end() + DOWN * 0.5)
109 truncated_label = Tex("\\textbf{Truncated SMN Protein}", font_size=28,
↳ color=ORANGE).next_to(truncated_protein, DOWN, buff=0.3)
110
111 self.play(
112     Create(arrow),
113     Write(truncated_label),
114     Create(truncated_protein),
115     run_time=1
116 )
117 self.wait(1)

```

```

118
119 class Scene19_NusinersenTherapy(Scene):
120     def construct(self):
121         # recreate the ending state of Scene 18
122         # title
123         title = Tex("\\textbf{SMN2 Gene As a Backup}", font_size=48, color=YELLOW)
124         title.move_to(UP * 3)
125         self.add(title)
126
127         # SMN2 RNA
128         x_start = -6
129         x_end = 6
130         amplitude = 0.3
131         frequency = 1
132         wave_func = lambda x: amplitude * np.sin(frequency * x)
133         smn2_rna = VGroup(
134             ParametricFunction(
135                 lambda t: np.array([t, wave_func(t) + 0.3, 0]),
136                 t_range=[x_start, x_end],
137                 color=BLUE,
138                 stroke_width=4
139             ),
140             ParametricFunction(
141                 lambda t: np.array([t, wave_func(t) - 0.3, 0]),
142                 t_range=[x_start, x_end],
143                 color=BLUE,
144                 stroke_width=4
145             )
146         )
147         smn2_rna.scale(1.3)
148         smn2_label = Tex("\\textbf{SMN2 RNA}", font_size=32, color=BLUE).next_to(smn2_rna, UP,
149 ↪ buff=0.5)
150         smn2_label.shift(LEFT * 1.7 + DOWN * 1)
151         self.add(smn2_rna, smn2_label)
152
153         # error marker
154         exclamation_mark = Tex("!", font_size=56, color=RED)
155         error_circle = Circle(radius=0.3, color=RED, stroke_width=3)
156         error_marker = VGroup(error_circle, exclamation_mark).move_to(smn2_rna.get_center() + RIGHT *
157 ↪ 2 + UP * 0.35)
158         self.add(error_marker)
159
160         # splice lines
161         splice_line1 = DashedLine(
162             start=smn2_rna[0].point_from_proportion(0.5),
163             end=smn2_rna[1].point_from_proportion(0.5),
164             color=WHITE,
165             stroke_width=4,
166             dash_length=0.06,
167             buff=0
168         )
169         splice_line2 = DashedLine(
170             start=smn2_rna[0].point_from_proportion(0.8),
171             end=smn2_rna[1].point_from_proportion(0.8),
172             color=WHITE,
173             stroke_width=4,
174             dash_length=0.06,
175             buff=0
176         )
177         exon7_text = Tex("\\textbf{Exon 7} \\\excluded", font_size=28,
178 ↪ color=RED).next_to(splice_line2, RIGHT, buff=0.3)
179         exon7_text.next_to(error_marker, DOWN, buff=0.2)
180         self.add(splice_line1, splice_line2, exon7_text)

```

```

179     # arrow
180     arrow = Arrow(
181         start=splice_line1.get_bottom() + DOWN * 0.2,
182         end=splice_line1.get_bottom() + DOWN * 1.2,
183         color=YELLOW,
184         buff=0,
185         stroke_width=6
186     )
187     self.add(arrows)
188
189     # truncated protein
190     truncated_protein = Arc(
191         radius=0.4,
192         start_angle=PI / 4,
193         angle=3 * PI / 2,
194         color=ORANGE,
195         stroke_width=6
196     ).move_to(arrows.get_end() + DOWN * 0.5)
197     truncated_label = Tex("\\textbf{Truncated SMN Protein}", font_size=28,
198         ↪ color=ORANGE).next_to(truncated_protein, DOWN, buff=0.3)
199     self.add(truncated_protein, truncated_label)
200
201     self.wait(1)
202
203     ##### NEW SCENE
204
205     new_title = Tex("\\textbf{Nusinersen Therapy}", font_size=48, color=YELLOW)
206     new_title.move_to(UP * 3)
207     self.play(
208         Transform(title, new_title),
209         FadeOut(exon7_text),
210         FadeOut(splice_line1), FadeOut(splice_line2),
211         run_time=1
212     )
213
214     # Proofreader (a purple oval)
215     proofreader = Ellipse(width=1.8, height=1.3, color=PURPLE, fill_color=PURPLE,
216         ↪ fill_opacity=0.5, stroke_width=3)
217     proofreader_label = Tex("\\textbf{Nusinersen}", font_size=32, color=WHITE)
218     proofreader_label.move_to(proofreader.get_center())
219     proofreader_group = VGroup(proofreader, proofreader_label)
220     # move out of screen on the right
221     proofreader_group.move_to(error_marker.get_center() + RIGHT * 8 + UP * 0.5)
222
223     # animate proofreader moving to the error site
224     self.play(
225         proofreader_group.animate.move_to(error_marker.get_center() + UP * 0.5),
226         run_time=1
227     )
228
229     # animate proofreader fixing the error
230     self.play(
231         FadeOut(error_marker),
232         Flash(error_marker.get_center(), color=GREEN, flash_radius=0.4, line_stroke_width=6),
233         run_time=1
234     )
235     self.wait(1)
236
237     # fixed protein
238     fixed_protein = Circle(radius=0.4, color=GREEN, stroke_width=6)
239     fixed_label = Tex("\\textbf{Full-length SMN Protein}", font_size=28, color=GREEN)
240     fixed_protein.move_to(truncated_protein.get_center())
241     fixed_label.move_to(truncated_label.get_center())
242
243     # animate splicing and protein change

```

```

241     self.play(
242         Transform(truncated_protein, fixed_protein),
243         Transform(truncated_label, fixed_label),
244         run_time=1
245     )
246
247     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.

```

1  from manim import *
2  import random
3
4  # ETHICAL DISCUSSION SCENES
5
6  class Scene20_EthicalTitle(Scene):
7      """Chapter Title: Ethical Considerations"""
8      def construct(self):
9          # Text animation
10         chapter = Tex("{\\scshape\\bfseries CHAPTER V}", font_size=50, color=YELLOW)
11         title = Tex("{\\bfseries Gene Therapy Ethics}", font_size=64, color=YELLOW)
12         separator = Line(start=LEFT * 3, end=RIGHT * 3, color=ORANGE, stroke_width=4)
13
14         chapter.move_to(UP * 0.75)
15         title.move_to(DOWN * 0.75)
16         separator.move_to(ORIGIN)
17
18         self.play(
19             Write(chapter, run_time=1.5),
20             Create(separator, run_time=1.5),
21             Write(title, run_time=1.5)
22         )
23
24         self.wait(3)
25
26  class Scene21_HumanWithQuestionMarks(Scene):
27      def construct(self):
28          # human svg
29         human = SVGObject("./human_outline.svg", fill_color=BLUE, fill_opacity=0.3,
30             ↪ stroke_color=BLUE, stroke_width=2, should_center=True)
31         human.scale(5)
32         human.move_to(DOWN*1.5)
33         self.play(Create(human), run_time=2)
34
35         # Create question marks with different properties
36         question_marks = []
37         colors = [RED, YELLOW, GREEN, PURPLE, ORANGE, PINK, TEAL]
38         positions = [
39             DOWN * 2 + LEFT * 3,
40             UP * 3 + RIGHT * 2,
41             LEFT * 4 + UP * 1,
42             RIGHT * 4 + UP * 0.5,
43             DOWN * 2.5 + RIGHT * 4,
44             LEFT * 3.5 + UP * 3,
45             RIGHT * 3 + DOWN * 3.5,
46             LEFT * 2 + UP * 1.5,
47             RIGHT * 2.5 + UP * 2,
48             LEFT * 5 + UP * 2.5,
49             RIGHT * 5 + UP * 1.5,

```

```

49         DOWN * 3.5 + LEFT * 1.5,
50     ]
51
52     # Predefined sizes for each question mark
53     sizes = [65, 72, 48, 55, 78, 43, 69, 51, 76, 58, 62, 71]
54
55     for i, (pos, color, size) in enumerate(zip(positions, colors * 2, sizes)):
56         qmark = Tex("?", font_size=size, color=color)
57         qmark.move_to(pos)
58         question_marks.append(qmark)
59
60     # Fade in question marks with lag
61     self.play(
62         LaggedStart(*[FadeIn(qmark) for qmark in question_marks],
63                     lag_ratio=0.5,
64                     run_time=6)
65     )
66
67     # text
68     philosophical_text = Tex("$\\bullet$ Philosophical question?", font_size=50, color=WHITE)
69     medical_safety_text = Tex("$\\bullet$ Medical safety!", font_size=50, color=WHITE)
70
71     philosophical_text.move_to(DOWN * 2.6 + LEFT * 4)
72     medical_safety_text.next_to(philosophical_text, DOWN, aligned_edge=LEFT, buff=0.2)
73
74     self.play(Write(philosophical_text, run_time=1))
75     self.wait(0.5)
76     self.play(Write(medical_safety_text, run_time=1))
77     self.wait(3)
78
79 class Scene22_JesseGelsingerCaseTimeline(Scene):
80     def construct(self):
81         # Title
82         title = Tex("\\textbf{Jesse Gelsinger Case Timeline}", font_size=48, color=YELLOW)
83         title.move_to(UP * 3.5)
84         self.play(Write(title, run_time=0.5))
85         self.wait(0.5)
86
87         # Timeline arrow
88         timeline = Arrow(start=LEFT * 6 + UP * 3, end=LEFT * 6 + DOWN * 4, color=YELLOW,
89                          ↪ stroke_width=4)
89         self.play(Create(timeline), run_time=0.5)
90         self.wait(0.5)
91
92         # Events
93         events = [
94             ("\\bfseries 18 Jun 1981:", "Jesse Gelsinger born, later diagnosed with OTC
95             ↪ deficiency"),
96             ("\\bfseries 13 Sept 1999:", "At 18, Gelsinger injected with experimental gene
97             ↪ therapy"),
98             ("\\bfseries 17 Sept 1999:", "Gelsinger died from organ failure due to immune
99             ↪ response"),
100         ]
101
102         for i, event_text in enumerate(events):
103             event = VGroup(
104                 Tex(event_text[0], font_size=28, color=WHITE),
105                 Tex(event_text[1], font_size=32, color=WHITE)
106             )
107             event.arrange(DOWN, aligned_edge=LEFT)
108             x_pos = LEFT * 6
109             y_pos = UP * 2 + (DOWN * 4) * (i / (len(events) - 1))
110             marker = Dot(point=x_pos + y_pos, color=RED, radius=0.1)
111             event.next_to(marker, RIGHT, buff=0.3)

```

```

109         self.play(Create(marker), Write(event, run_time=1))
110         self.wait(0.5)
111
112     self.wait(3)
113
114 class Scene23_BubbleBoy(Scene):
115     def construct(self):
116         # Create 4 stick figures in blue bubbles arranged in 2x2 grid
117         stick_figures = []
118         bubbles = []
119
120         # Positions for 2x2 grid
121         positions = [
122             UP * 1.5 + LEFT * 2, # Top left
123             UP * 1.5 + RIGHT * 2, # Top right
124             DOWN * 1.5 + LEFT * 2, # Bottom left
125             DOWN * 1.5 + RIGHT * 2 # Bottom right
126         ]
127
128         for pos in positions:
129             # Create stick figure
130             stick = VGroup(
131                 Circle(radius=0.15, color=WHITE, fill_opacity=1), # Head
132                 Line(start=UP * 0.25, end=DOWN * 0.5, color=WHITE), # Body
133                 Line(start=UP * 0.25, end=DOWN * 0.15 + LEFT * 0.25, color=WHITE), # Left arm
134                 Line(start=UP * 0.25, end=DOWN * 0.15 + RIGHT * 0.25, color=WHITE), # Right arm
135                 Line(start=DOWN * 0.5, end=DOWN * 0.8 + LEFT * 0.2, color=WHITE), # Left leg
136                 Line(start=DOWN * 0.5, end=DOWN * 0.8 + RIGHT * 0.2, color=WHITE), # Right leg
137             )
138             stick[0].move_to(UP * 0.4)
139             stick.scale(0.8)
140             stick.move_to(pos)
141
142             # Create bubble
143             bubble = Circle(radius=1.2, color=BLUE, stroke_width=4, fill_opacity=0.3, fill_color=BLUE)
144             bubble.move_to(pos)
145
146             stick_figures.append(stick)
147             bubbles.append(bubble)
148
149         # Show all stick figures and blue bubbles
150         self.play(
151             *[Create(bubble) for bubble in bubbles],
152             *[Create(stick) for stick in stick_figures],
153             run_time=1
154         )
155         self.wait(0.5)
156
157         # Change colors: bottom right to red, others to green
158         # Also create warning text for bottom right
159         warning_text = Tex("Accidental activation\\\\" of leukaemia (25\\%)", font_size=32, color=RED)
160         warning_text.next_to(bubbles[3], DOWN, buff=0.2)
161
162         self.play(
163             bubbles[3].animate.set_color(RED), # Bottom right to red
164             *[bubbles[i].animate.set_color(GREEN) for i in range(3)], # Others to green
165             Write(warning_text),
166             run_time=1
167         )
168
169         self.wait(3)
170
171 class Scene24_ImmuneToGeneTherapy(Scene):

```

```

172 """Scene showing immune response to gene therapy"""
173 def construct(self):
174     # 1. Create green human outline on the right half
175     human = SVGObject("./human_outline.svg", fill_color=GREEN, fill_opacity=0.3,
176 ↪ stroke_color=GREEN, stroke_width=3, should_center=True)
177     human.scale(3)
178     human.move_to(RIGHT * 3)
179
180     # Create AAV viral vector with RNA strand (upper-left)
181     aav_capsid = RegularPolygon(n=6, radius=1.2, color=PURPLE, stroke_width=4, fill_color=PURPLE,
182 ↪ fill_opacity=0.2)
183
184     spike_positions = [
185         UP * 1.2,
186         UP * 0.6 + RIGHT * 1.0,
187         DOWN * 0.6 + RIGHT * 1.0,
188         DOWN * 1.2,
189         DOWN * 0.6 + LEFT * 1.0,
190         UP * 0.6 + LEFT * 1.0
191     ]
192
193     spikes = VGroup()
194     for pos in spike_positions:
195         spike = Triangle(color=PURPLE, fill_color=PURPLE, fill_opacity=0.8)
196         spike.scale(0.15)
197         spike.move_to(pos)
198         angle = np.arctan2(pos[1], pos[0])
199         spike.rotate(angle + PI/2)
200         spikes.add(spike)
201
202     # RNA strand inside
203     rna_strand = ParametricFunction(
204         lambda t: np.array([
205             0.2 * np.sin(4 * PI * t),
206             1.5 * t - 0.75,
207             0
208         ]),
209         t_range=[0, 1],
210         color=GREEN,
211         stroke_width=6
212     ).scale(0.3)
213
214     aav_capsid.scale(0.5)
215     spikes.scale(0.5)
216     aav_vector = VGroup(aav_capsid, spikes, rna_strand)
217     aav_vector.move_to(UP * 2 + LEFT * 4)
218
219     # Create CRISPR-Cas9 protein (lower-left)
220     crispr_cas9 = VGroup(
221         Rectangle(width=1.5, height=0.6, color=BLUE, fill_color=BLUE, fill_opacity=0.5),
222         Tex("\\textbf{CRISPR-\\Cas9}", font_size=20, color=BLUE)
223     )
224     crispr_cas9.scale(1.4)
225     crispr_cas9[1].move_to(crispr_cas9[0].get_center())
226     crispr_cas9.move_to(DOWN * 2 + LEFT * 4)
227
228     # Display all elements
229     self.play(
230         Create(human),
231         Create(aav_vector),
232         Create(crispr_cas9),
233         run_time=1
234     )
235     self.wait(1)

```

```

234
235 # 2. Vector approaches human but is bounced back
236 target_pos = human.get_left() + LEFT * 0.5
237 self.play(
238     aav_vector.animate.move_to(target_pos),
239     run_time=1
240 )
241 # self.wait(0.3)
242
243 # Bounce back with shake
244 self.play(
245     aav_vector.animate.shift(LEFT * 0.3),
246     run_time=0.2
247 )
248 # self.play(
249 #     aav_vector.animate.shift(LEFT * 1.5),
250 #     run_time=0.8,
251 #     rate_func=smooth
252 # )
253 self.wait(0.5)
254
255 # 3. Red cross forms on vector and it turns gray
256 cross_line1 = Line(
257     start=UP * 0.5 + LEFT * 0.5,
258     end=DOWN * 0.5 + RIGHT * 0.5,
259     color=RED,
260     stroke_width=8
261 )
262 cross_line2 = Line(
263     start=UP * 0.5 + RIGHT * 0.5,
264     end=DOWN * 0.5 + LEFT * 0.5,
265     color=RED,
266     stroke_width=8
267 )
268 red_cross_vector = VGroup(cross_line1, cross_line2)
269 red_cross_vector.move_to(aav_vector.get_center())
270
271 self.play(
272     Create(red_cross_vector),
273     aav_capsid.animate.set_color(GRAY),
274     aav_capsid.animate.set_fill(GRAY, opacity=0.2),
275     *[spike.animate.set_color(GRAY).set_fill(GRAY, opacity=0.2) for spike in spikes],
276     aav_vector.animate.set_stroke(GRAY),
277     rna_strand.animate.set_color(GRAY),
278     run_time=1
279 )
280 self.wait(1)
281
282 # 4. CRISPR-Cas9 enters the human
283 self.play(
284     crispr_cas9.animate.move_to(human.get_center()),
285     run_time=2
286 )
287 self.wait(0.5)
288
289 # Cross forms on human and human turns gray
290 cross_line3 = Line(
291     start=UP * 1.5 + LEFT * 1.5,
292     end=DOWN * 1.5 + RIGHT * 1.5,
293     color=RED,
294     stroke_width=10
295 )
296 cross_line4 = Line(

```

```

297         start=UP * 1.5 + RIGHT * 1.5,
298         end=DOWN * 1.5 + LEFT * 1.5,
299         color=RED,
300         stroke_width=10
301     )
302     red_cross_human = VGroup(cross_line3, cross_line4)
303     red_cross_human.move_to(human.get_center())
304
305     self.play(
306         Create(red_cross_human),
307         human.animate.set_color(GRAY).set_fill(GRAY, opacity=0.3),
308         crispr_cas9.animate.set_color(GRAY).set_opacity(0.3),
309         run_time=1.5
310     )
311     self.wait(2)
312
313 class Scene25_PublicOpinionSurvey(Scene):
314     """Public opinion survey on gene therapy"""
315     def construct(self):
316         # Title
317         title = Tex("\\textbf{Public Opinion Survey}", font_size=48, color=YELLOW)
318         title.move_to(UP * 3.2)
319
320         # Create 100 people icons in 10x10 grid
321         people_icons = VGroup()
322         rows, cols = 10, 10
323         spacing = 0.6
324
325         for row in range(rows):
326             for col in range(cols):
327                 person = Circle(radius=0.08, color=WHITE, fill_opacity=1, fill_color=WHITE)
328                 person[0].move_to(UP * 0.1)
329
330                 x_pos = (col - cols / 2) * spacing + spacing / 2
331                 y_pos = (rows / 2 - row) * spacing - spacing / 2
332                 person.move_to(RIGHT * x_pos + UP * y_pos + DOWN * 0.5)
333
334                 people_icons.add(person)
335
336         self.play(
337             Write(title),
338             LaggedStart(*[FadeIn(person) for person in people_icons], lag_ratio=0.01),
339             run_time=1
340         )
341         self.wait(1)
342
343         # Group them randomly (predefined random)
344         oppose_indices = (
345             2, 5, 8, 13, 14, 19, 20, 23, 27, 28,
346             30, 42, 45, 47, 50, 51, 52, 60, 63, 68,
347             69, 94, 97, 99, 92
348         )
349         oppose_people = [people_icons[i] for i in oppose_indices]
350         support_people = [people_icons[i] for i in range(100) if i not in oppose_indices]
351
352         # Highlight 75% in green (supporting gene therapy)
353         support_label = Tex("\\textbf{75\\% believe in\\gene therapy}", font_size=48, color=GREEN)
354         support_label.move_to(LEFT * 5)
355
356         self.play(
357             *[support_people[i].animate.set_color(GREEN) for i in range(75)],
358             Write(support_label),
359             run_time=1

```

```

360     )
361     self.wait(0.5)
362
363     # Show the remaining 25% in red
364     oppose_label = Tex("\\textbf{25\\% uncertain\\or opposed}", font_size=48, color=RED)
365     oppose_label.move_to(RIGHT * 5)
366
367     self.play(
368         *[oppose_people[i].animate.set_color(RED) for i in range(25)],
369         Write(oppose_label),
370         run_time=1
371     )
372
373     self.wait(3)
374
375 class Scene26_EthicalConcerns(Scene):
376     """Ethical concerns about gene therapy"""
377     def construct(self):
378         # Title
379         title = Tex("\\bfseries Ethical Concerns", font_size=56, color=YELLOW)
380         title.move_to(UP * 3)
381
382         # Concern statements
383         concern1_title = Tex("$\\bullet$ \\textbf{`Playing God'})", font_size=36, color=RED)
384         concern1_detail = Tex("Changing nature may cause unexpected dangerous results", font_size=28)
385         concern1 = VGroup(concern1_title, concern1_detail).arrange(DOWN, buff=0.2, aligned_edge=LEFT)
386
387         concern2_title = Tex("$\\bullet$ \\textbf{Fairness and Access}", font_size=36, color=ORANGE)
388         concern2_detail = Tex("Treatments cost millions, risking a permanent genetic divide",
389                               ↪ font_size=28)
390         concern2 = VGroup(concern2_title, concern2_detail).arrange(DOWN, buff=0.2, aligned_edge=LEFT)
391
392         concern3_title = Tex("$\\bullet$ \\textbf{Threat to Human Identity}", font_size=36,
393                               ↪ color=PURPLE)
394         concern3_detail = Tex("Brain-editing errors could cause catastrophic damage", font_size=28)
395         concern3 = VGroup(concern3_title, concern3_detail).arrange(DOWN, buff=0.2, aligned_edge=LEFT)
396
397         concerns = VGroup(concern1, concern2, concern3).arrange(DOWN, buff=0.7, aligned_edge=LEFT)
398         concerns.move_to(DOWN * 0.3)
399
400         # Animate title
401         self.play(Write(title), run_time=1)
402         self.wait(0.5)
403
404         # Animate concerns appearing one by one
405         self.play(FadeIn(concern1, shift=UP * 0.3), run_time=1)
406         self.wait(0.5)
407         self.play(FadeIn(concern2, shift=UP * 0.3), run_time=1)
408         self.wait(0.5)
409         self.play(FadeIn(concern3, shift=UP * 0.3), run_time=1)
410
411         self.wait(3)

```