Abstraction by the rule of 10

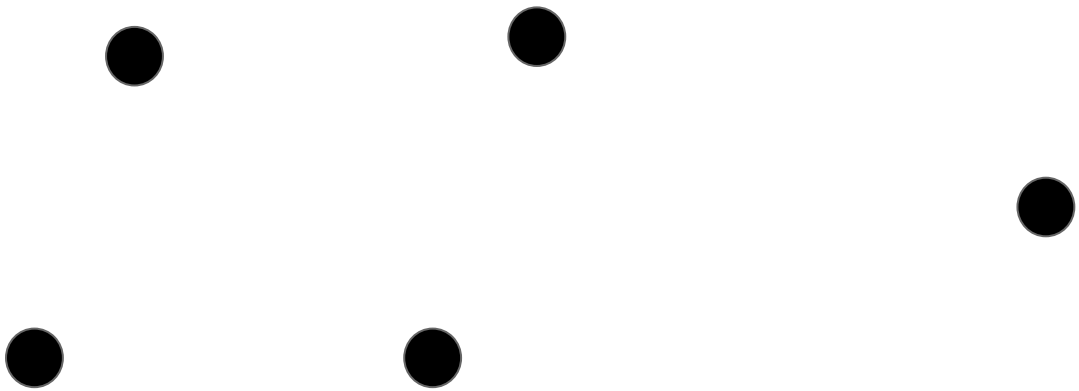
Guy Davidson

Meeting C++ 15/11/2019

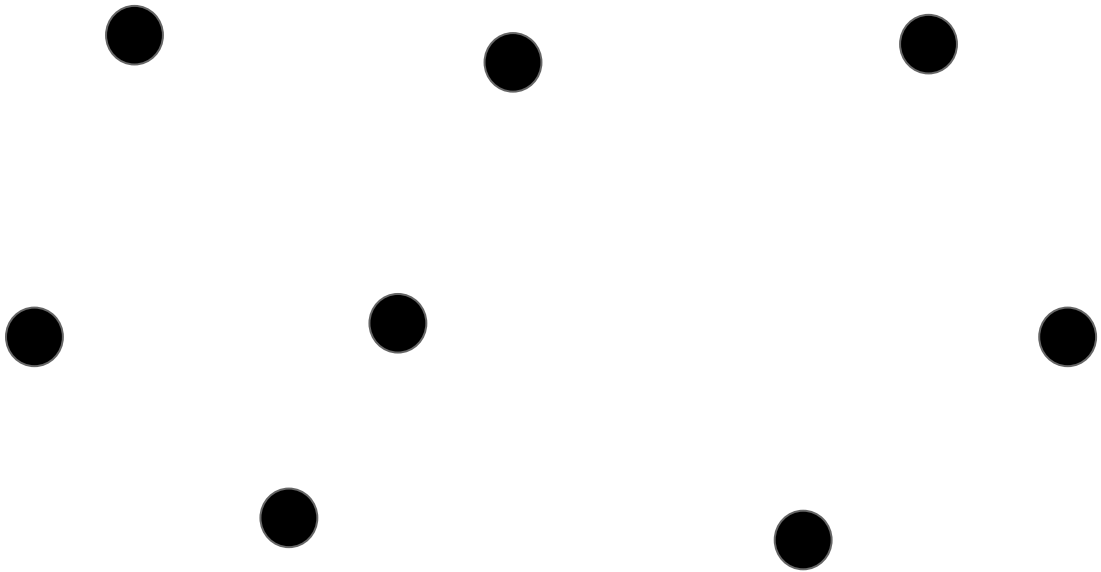
@hatcat01

Good evening Guy

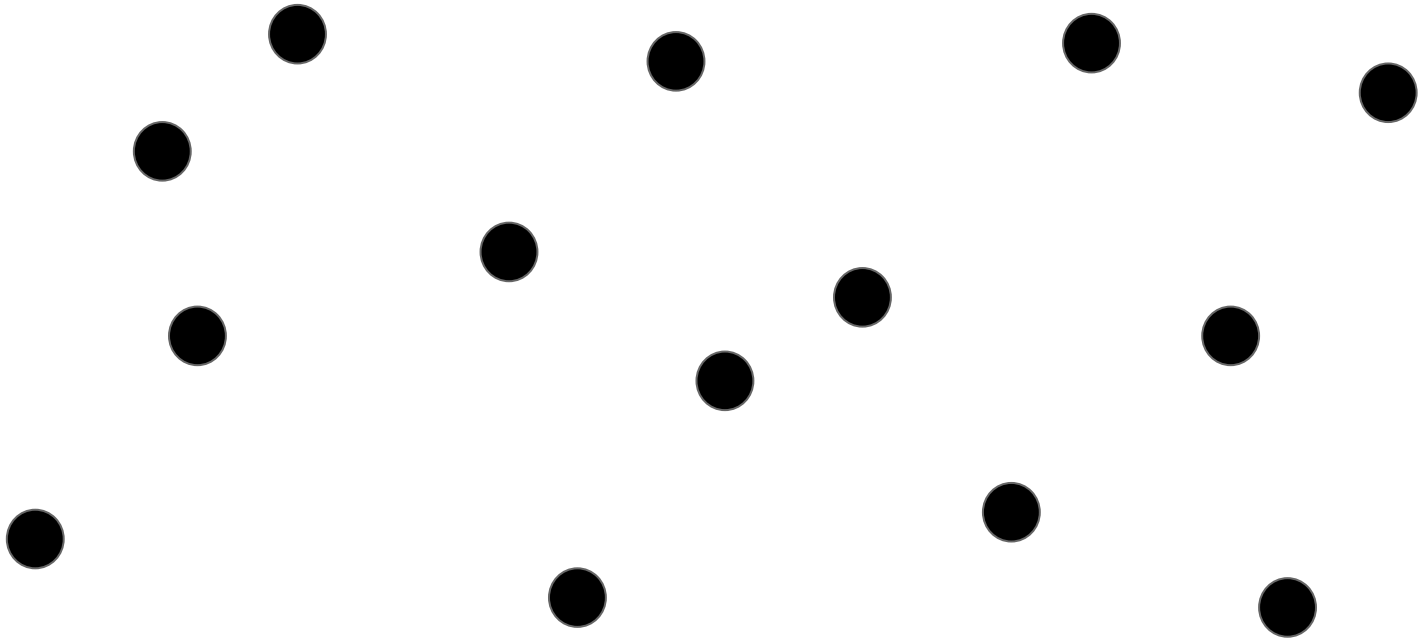
How many dots on this slide?



How many dots on this slide?



How many dots on this slide?



Immediate apprehension

Limit of about 10

Immediate apprehension

Limit of about 10

More is too big to swallow in one gulp

Immediate apprehension

Limit of about 10

More is too big to swallow in one gulp

Smaller load => less friction

C++ Abstractions

C++ Abstractions

Memory => identifiers

C++ Abstractions

```
Memory => identifiers

Memory => identifiers
```

C++ Abstractions

```
void bezier_animation::render(unmanaged_output_surface& uos)
{
	auto time_in_slide = std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::steady_clock::now() - m_entry_point).count();
	m_bg.render(uos);
	auto left = path_builder{};
	left.clear();
	left.new_figure(point_2d{ 350.f, 825.f });
	left.line(point_2d{ 350.f, 255.f });
	left.line(point_2d{ 925.f, 825.f });
	auto right = path_builder{};
	right.clear();
	right.new_figure(point_2d{ 995.f, 825.f });
	right.line(point_2d{ 995.f, 255.f });
	right.line(point_2d{ 1565.f, 255.f });
	auto left_curve = path_builder{};
	left_curve.clear();
	left_curve.new_figure(point_2d{ 350.f, 825.f });
	left_curve.quadratic_curve(point_2d{ 350.f, 255.f }, point_2d{ 925.f, 825.f });
	auto right_curve = path_builder{};
	right_curve.clear();
	right_curve.new_figure(point_2d{ 995.f, 825.f });
	right_curve.quadratic_curve(point_2d{ 995.f, 255.f }, point_2d{ 1565.f, 255.f });
	if (time_in_slide <= 5000)
	{
		auto fraction = (time_in_slide) / 5000.f;
		auto l_x1 = 350.f;
		auto l_y1 = show::delta(825.f, 255.f, fraction);
		auto l_x2 = show::delta(350.f, 925.f, fraction);
		auto l_y2 = show::delta(255.f, 825.f, fraction);
		auto r_x1 = 995.f;
		auto r_y1 = show::delta(825.f, 255.f, fraction);
		auto r_x2 = show::delta(995.f, 1565.f, fraction);
		auto r_y2 = 255.f;
		auto left_normal = path_builder{};
		left_normal.clear();
		left_normal.new_figure({ l_x1, l_y1 });
		left_normal.line({ l_x2, l_y2 });
```

C++ Abstractions

```
void bezier_animation::render(unmanaged_output_surface& uos)
{
}
```

C++ Abstractions

```
void
bezier_animation::render_a_curve
(unmanaged_output_surface& uos, curve& c)
{...}

void
bezier_animation::render_all_the_curves
(unmanaged_output_surface& uos)
{...}
```

C++ Abstractions

```
class curves
{
public:
    void render(unmanaged_output_surface&) const;
    void add_curve(p_2d begin, p_2d control, p_2d end);
private:
    std::vector<curve>          the_curves;
    unmanaged_output_surface    surface;
};
```

C++ Abstractions

```
class bezier_animation : public animation
{
public:
  bezier_animation(std::vector<bezier>&& curves, std::vector<shape>&& shapes);
  void tweak_control_point(int idx, float x_delta, float y_delta);
  void normalise(int idx_1, int idx_2, float q1, float q2, float norm, float delta, float pain_point_1, float pain_point_2 = 37.356f);
  void redistribute(bool x);
  void gather(bool x, bool align, bool centre);
  void force(bool c, bool d, float range_range);
  void enumerate_controls(enum_control_cb) const;
  void enumerate_shapes(enum_shape_cb cb) const;
  float control_point_separation(int idx_1, int idx_2) const;
  float shape_distance_avg(int idx_1, int idx_2) const;
// animation overrides
  void update_all() override;
  void pause() override;
  animation& next_target() override;
  void animate(int x, float interval) override;
  void rewind(int idx, float interval) override;
  void advance(int idx, float interval) override;
  bool complete(int idx) const override;
  bool animating(int idx) const override;
  float remaining_interval(int idx) const override;
  float end_point_delta(int begin, int end) const override;
private:
  std::vector<bezier> m_curves;
  std::vector<shape> m_shapes;
  std::vector<std::pair<float, float>> m_control_points;
  std::vector<float> m_intervals;
  std::pair<std::pair<float, float>, std::pair<float, float>> m_end_points;
  int m_index_count;
  std::vector<controllers> m_controllers;
  int m_active_index;
  int m_prior_index;
  int m_next_index;
  int m_active_control;
  int m_next_control;
```

C++ Abstractions

```
class bezier_animation : public animation
{
public:
  bezier_animation(std::vector<bezier>&& curves, std::vector<shape>&& shapes);
  void tweak_control_point(int idx, float x_delta, float y_delta);
  void normalise(int idx_1, int idx_2, float q1, float q2, float norm, float delta, float pain_point_1, float pain_point_2 = 37.356f);
  void redistribute(bool x);
  void gather(bool x, bool align, bool centre);
  void force(bool c, bool d, float range_range);
  void enumerate_controls(enum_control_cb) const;
  void enumerate_shapes(enum_shape_cb cb) const;
  float control_point_separation(int idx_1, int idx_2) const;
  float shape_distance_avg(int idx_1, int idx_2) const;
// animation overrides
  void update_all() override;
  void pause() override;
  animation& next_target() override;
  void animate(int x, float interval) override;
  void rewind(int idx, float interval) override;
  void advance(int idx, float interval) override;
  bool complete(int idx) const override;
  bool animating(int idx) const override;
  float remaining_interval(int idx) const override;
  float end_point_delta(int begin, int end) const override;
private:
  std::vector<bezier> m_curves;
  std::vector<shape> m_shapes;
  std::vector<std::pair<float, float>> m_control_points;
  std::vector<float> m_intervals;
  std::pair<std::pair<float, float>, std::pair<float, float>> m_end_points;
  int m_index_count;
  std::vector<controllers> m_controllers;
  int m_active_index;
  int m_prior_index;
  int m_next_index;
  int m_active_control;
  int m_next_control
```

C++ Abstractions

```
class bezier_animation : public animation
{
public:
  bezier_animation(std::vector<bezier>&& curves, std::vector<shape>&& shapes);
  void tweak_control_point(int idx, float x_delta, float y_delta);
  void normalise(int idx_1, int idx_2, float q1, float q2, float norm, float delta, float pain_point_1, float pain_point_2 = 37.356f);
  void redistribute(bool x);
  void gather(bool x, bool align, bool centre);
  void force(bool c, bool d, float range_range);
  void enumerate_controls(enum_control_cb) const;
  void enumerate_shapes(enum_shape_cb cb) const;
  float control_point_separation(int idx_1, int idx_2) const;
  float shape_distance_avg(int idx_1, int idx_2) const;
// animation overrides
  void update_all() override;
  void pause() override;
  animation& next_target() override;
  void animate(int x, float interval) override;
  void rewind(int idx, float interval) override;
  void advance(int idx, float interval) override;
  bool complete(int idx) const override;
  bool animating(int idx) const override;
  float remaining_interval(int idx) const override;
  float end_point_delta(int begin, int end) const override;
private:
  std::vector<bezier> m_curves;
  std::vector<shape> m_shapes;
  std::vector<std::pair<float, float>> m_control_points;
  std::vector<float> m_intervals;
  std::pair<std::pair<float, float>, std::pair<float, float>> m_end_points;
  int m_index_count;
  std::vector<controllers> m_controllers;
  int m_active_index;
  int m_prior_index;
  int m_next_index;
  int m_active_control;
  int m_next_control;
```

C++ Abstractions

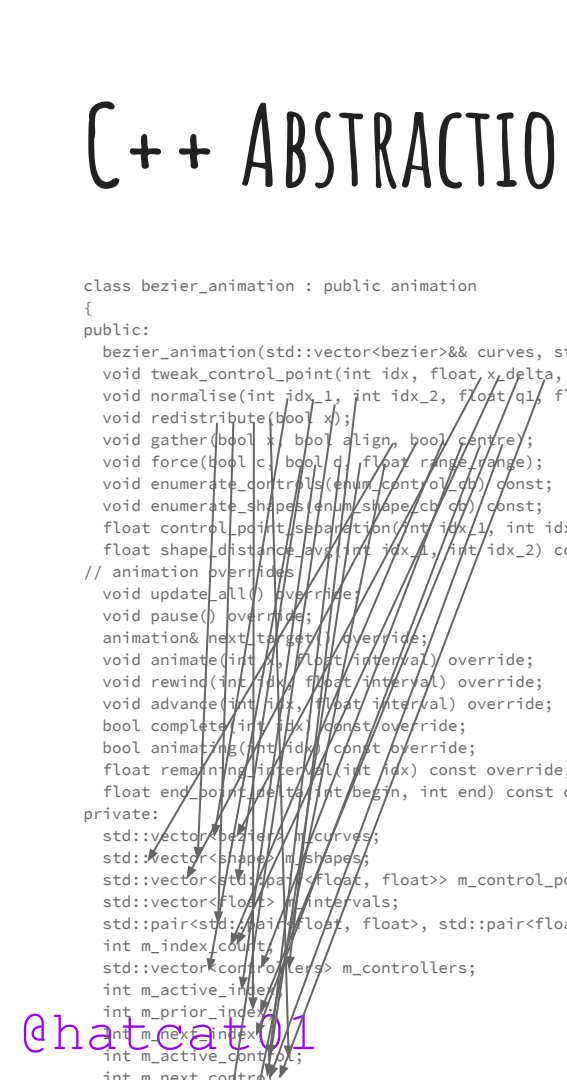
```
class bezier_animation : public animation
{
public:
  bezier_animation(std::vector<bezier>&& curves, std::vector<shape>&& shapes);
  void tweak_control_point(int idx, float x_delta, float y_delta);
  void normalise(int idx_1, int idx_2, float q1, float q2, float norm, float delta, float pain_point_1, float pain_point_2 = 37.356f);
  void redistribute(bool x);
  void gather(bool x, bool align, bool centre);
  void force(bool c, bool d, float range_range);
  void enumerate_controls(enum_control_cb) const;
  void enumerate_shapes(enum_shape_cb cb) const;
  float control_point_separation(int idx_1, int idx_2) const;
  float shape_distance_avg(int idx_1, int idx_2) const;
// animation overrides
  void update_all() override;
  void pause() override;
  animation& next_target() override;
  void animate(int x, float interval) override;
  void rewind(int idx, float interval) override;
  void advance(int idx, float interval) override;
  bool complete(int idx) const override;
  bool animating(int idx) const override;
  float remaining_interval(int idx) const override;
  float end_point_delta(int begin, int end) const override;
private:
  std::vector<bezier> m_curves;
  std::vector<shape> m_shapes;
  std::vector<std::pair<float, float>> m_control_points;
  std::vector<float> m_intervals;
  std::pair<std::pair<float, float>, std::pair<float, float>> m_end_points;
  int m_index_count;
  std::vector<controllers> m_controllers;
  int m_active_index;
  int m_prior_index;
  int m_next_index;
  int m_active_control;
  int m_next_control;
```

C++ Abstractions

```
class bezier_animation : public animation
{
public:
  bezier_animation(std::vector<bezier>&& curves, std::vector<shape>&& shapes);
  void tweak_control_point(int idx, float x_delta, float y_delta);
  void normalise(int idx_1, int idx_2, float q1, float q2, float norm, float delta, float pain_point_1, float pain_point_2 = 37.356f);
  void redistribute(bool x);
  void gather(bool x, bool align, bool centre);
  void force(bool c, bool d, float range_range);
  void enumerate_controls(enum_control_cb cb) const;
  void enumerate_shapes(enum_shape_cb cb) const;
  float control_point_separation(int idx_1, int idx_2) const;
  float shape_distance_avg(int idx_1, int idx_2) const;
// animation overrides
  void update_all() override;
  void pause() override;
  animation& next_target() override;
  void animate(int idx, float interval) override;
  void rewind(int idx, float interval) override;
  void advance(int idx, float interval) override;
  bool complete(int idx) const override;
  bool animating(int idx) const override;
  float remaining_interval(int idx) const override;
  float end_point_delta(int begin, int end) const override;
private:
  std::vector<bezier> m_curves;
  std::vector<shape> m_shapes;
  std::vector<std::pair<float, float>> m_control_points;
  std::vector<float> m_intervals;
  std::pair<std::pair<float, float>, std::pair<float, float>> m_end_points;
  int m_index_count;
  std::vector<controllers> m_controllers;
  int m_active_index;
  int m_prior_index;
  int m_next_index;
  int m_active_control;
  int m_next_control;
```

C++ Abstractions

```
class bezier_animation : public animation
{
public:
  bezier_animation(std::vector<bezier>&& curves, std::vector<shape>&& shapes);
  void tweak_control_point(int idx, float x_delta, float y_delta);
  void normalise(int idx_1, int idx_2, float q1, float q2, float norm, float delta, float pain_point_1, float pain_point_2 = 37.356f);
  void redistribute(bool x);
  void gather(bool x, bool align, bool centre);
  void force(bool c, bool d, float range_range);
  void enumerate_controls(enum_control_cb cb) const;
  void enumerate_shapes(enum_shape_cb cb) const;
  float control_point_separation(int idx_1, int idx_2) const;
  float shape_distance_avg(int idx_1, int idx_2) const;
// animation overrides
  void update_all() override;
  void pause() override;
  animation& next_target() override;
  void animate(int idx, float interval) override;
  void rewind(int idx, float interval) override;
  void advance(int idx, float interval) override;
  bool complete(int idx) const override;
  bool animating(int idx) const override;
  float remaining_interval(int idx) const override;
  float end_point_delta(int begin, int end) const override;
private:
  std::vector<bezier> m_curves;
  std::vector<shape> m_shapes;
  std::vector<std::pair<float, float>> m_control_points;
  std::vector<float> m_intervals;
  std::pair<std::pair<float, float>, std::pair<float, float>> m_end_points;
  int m_index_count;
  std::vector<controllers> m_controllers;
  int m_active_index;
  int m_prior_index;
  int m_next_index;
  int m_active_control;
  int m_next_control;
```



@hatcat01

C++ Abstractions

```
class bezier_animation : public animation
{
public:
  bezier_animation(std::vector<bezier>&& curves, std::vector<shape>&& shapes);
  void tweak_control_point(int idx, float x_delta, float y_delta);
  void normalise(int idx_1, int idx_2, float q1, float q2, float norm, float delta, float pain_point_1, float pain_point_2 = 37.356f);
  void redistribute(bool x);
  void gather(bool x, bool align, bool centre);
  void force(bool c, bool d, float range_range);
  void enumerate_controls(enum_control_cb cb) const;
  void enumerate_shapes(enum_shape_cb cb) const;
  float control_point_separation(int idx_1, int idx_2) const;
  float shape_distance_avg(int idx_1, int idx_2) const;
// animation overrides
  void update_all() override;
  void pause() override;
  animation& next_target() override;
  void animate(int idx, float interval) override;
  void rewind(int idx, float interval) override;
  void advance(int idx, float interval) override;
  bool complete(int idx) const override;
  bool animating(int idx) const override;
  float remaining_interval(int idx) const override;
  float end_point_delta(int begin, int end) const override;
private:
  std::vector<bezier> m_curves;
  std::vector<shape> m_shapes;
  std::vector<std::pair<float, float>> m_control_points;
  std::vector<float> m_intervals;
  std::pair<std::pair<float, float>, std::pair<float, float>> m_end_points;
  int m_index_count;
  std::vector<controller> m_controllers;
  int m_active_index;
  int m_prior_index;
  int m_next_index;
  int m_active_control;
  int m_next_control;
```

The lines!!!

C++ Abstractions

```
class bezier_animation : public animation
{
public:
  bezier_animation(std::vector<bezier>&& curves, std::vector<shape>&& shapes);
  void tweak_control_point(int idx, float x_delta, float y_delta);
  void normalise(int idx_1, int idx_2, float q1, float q2, float norm, float delta, float pain_point_1, float pain_point_2 = 37.356f);
  void redistribute(bool x);
  void gather(bool x, bool align, bool centre);
  void force(bool c, bool d, float range_range);
  void enumerate_controls(enum_control_cb cb) const;
  void enumerate_shapes(enum_shape_cb cb) const;
  float control_point_separation(int idx_1, int idx_2) const;
  float shape_distance_avg(int idx_1, int idx_2) const;
// animation overrides
  void update_all() override;
  void pause() override;
  animation& next_target() override;
  void animate(int idx, float interval) override;
  void rewind(int idx, float interval) override;
  void advance(int idx, float interval) override;
  bool complete(int idx) const override;
  bool animating(int idx) const override;
  float remaining_interval(int idx) const override;
  float end_point_delta(int begin, int end) const override;
private:
  std::vector<bezier> m_curves;
  std::vector<shape> m_shapes;
  std::vector<std::pair<float, float>> m_control_points;
  std::vector<float> m_intervals;
  std::pair<std::pair<float, float>, std::pair<float, float>> m_end_points;
  int m_index_count;
  std::vector<controller> m_controllers;
  int m_active_index;
  int m_prior_index;
  int m_next_index;
  int m_active_control;
  int m_next_control;
```

The lines!!!

The many-angled ones!!!

@hatcat01

C++ Abstractions

```
class bezier_animation : public animation
{
public:
  bezier_animation(std::vector<bezier>&& curves, std::vector<shape>&& shapes);
  void tweak_control_point(int idx, float x_delta, float y_delta);
  void normalise(int idx_1, int idx_2, float q1, float q2, float norm, float delta, float pain_point_1, float pain_point_2 = 37.356f);
  void redistribute(bool x);
  void gather(bool x, bool align, bool centre);
  void force(bool c, bool d, float range_range);
  void enumerate_controls(enum_control_cb cb) const;
  void enumerate_shapes(enum_shape_cb cb) const;
  float control_point_separation(int idx_1, int idx_2) const;
  float shape_distance_avg(int idx_1, int idx_2) const;
// animation overrides
  void update_all() override;
  void pause() override;
  animation& next_target() override;
  void animate(int idx, float interval) override;
  void rewind(int idx, float interval) override;
  void advance(int idx, float interval) override;
  bool complete(int idx) const override;
  bool animating(int idx) const override;
  float remaining_interval(int idx) const override;
  float end_point_delta(int begin, int end) const override;
private:
  std::vector<bezier> m_curves;
  std::vector<shape> m_shapes;
  std::vector<std::pair<float, float>> m_control_points;
  std::vector<float> m_intervals;
  std::pair<std::pair<float, float>, std::pair<float, float>> m_end_points;
  int m_index_count;
  std::vector<controller> m_controllers;
  int m_active_index;
  int m_prior_index;
  int m_next_index;
  int m_active_control;
  int m_next_control;
```

The lines!!!

The many-angled ones!!!

They are rising!!!

C++ Abstractions

```
class bezier_animation : public animation
{
public:
  bezier_animation(std::vector<bezier>&& curves, std::vector<shape>&& shapes);
  void tweak_control_point(int idx, float x_delta, float y_delta);
  void normalise(int idx_1, int idx_2, float q1, float q2, float norm, float delta, float pain_point_1, float pain_point_2 = 37.356f);
  void redistribute(bool x);
  void gather(bool x, bool align, bool centre);
  void force(bool c, bool d, float range_range);
  void enumerate_controls(enum_control_cb cb) const;
  void enumerate_shapes(enum_shape_cb cb) const;
  float control_point_separation(int idx_1, int idx_2) const;
  float shape_distance_avg(int idx_1, int idx_2) const;
// animation overrides
  void update_all() override;
  void pause() override;
  animation& next_target() override;
  void animate(int idx, float interval) override;
  void rewind(int idx, float interval) override;
  void advance(int idx, float interval) override;
  bool complete(int idx) const override;
  bool animating(int idx) const override;
  float remaining_interval(int idx) const override;
  float end_point_delta(int begin, int end) const override;
private:
  std::vector<bezier> m_curves;
  std::vector<shape> m_shapes;
  std::vector<std::pair<float, float>> m_control_points;
  std::vector<float> m_intervals;
  std::pair<std::pair<float, float>, std::pair<float, float>> m_end_points;
  int m_index_count;
  std::vector<controllers> m_controllers;
  int m_active_index;
  int m_prior_index;
  int m_next_index;
  int m_active_control;
  int m_next_control;
```

The lines!!!

The many-angled ones!!!

They are rising!!!

Obey Cthulhu!!!

NAMING

NAMING

Identifying is easy

NAMING

Identifying is easy

Naming is hard

NAMING

Identifying is easy

Naming is hard

Name it or suffer

Abstraction mechanisms

Object

Abstraction mechanisms

Object

Function

Abstraction mechanisms

Object

Function

Class

Abstraction mechanisms

Object

Function

Class

Namespace

Animation namespace

```
namespace animation {
  class base;
  class bezier_animation;
  class animation_timer;
  class parameteric_animator;
  class animation_path;
  class object_animation;
  class colour_animation;
  class parabolic_interpolator;
  class linear_interpolator;
  class slerp;
  class colour_animator;
  class separation_modifier;
  class animation_factory;
  class debug_animation_analyser;
  class reference_frame;
  class nurbs;
  class spline;
  class bezier_patch_deformer;
  class height_map_deformer;
  class edge_deformer;
  class vertex_animator;
  class discrete_volumetric_modifier;
  class blingulator;
  class procedural_texture_animator;
  class mighty_space_being_of_doom;
  class opening_credits;
  class particle_explosion;
  class attenuation_amplifier;
  class easter_bunny_goes_nuts;
  class that_dancing_baby;
  class here_is_johnny;
  class all_work_and_no_play;
  class morning_star;
  class flyaway;
  class asteroid;
  class cat_collision;
```

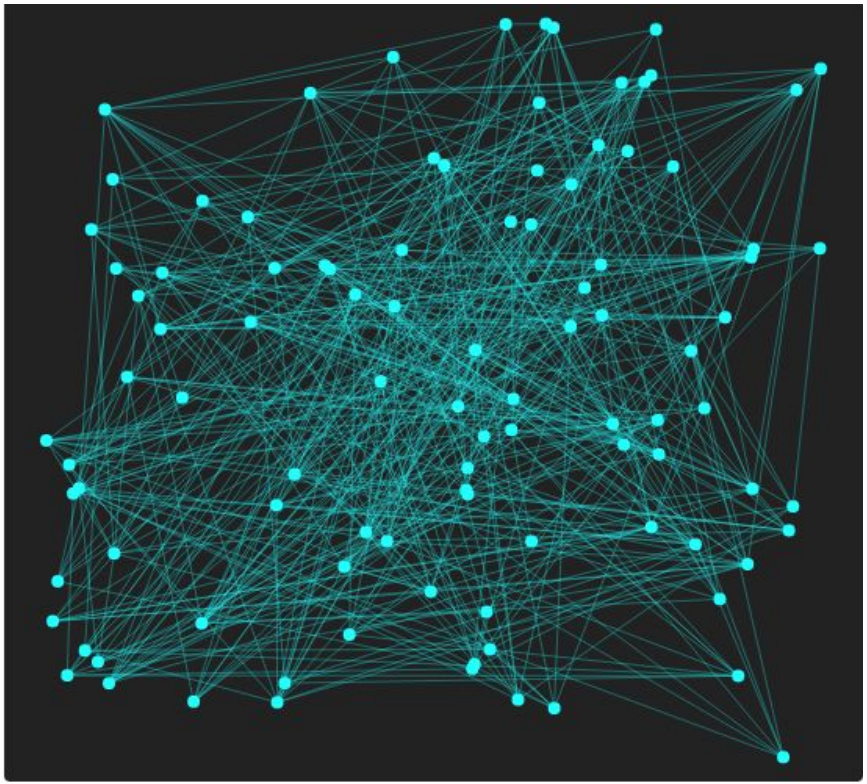
Animation namespace

```
namespace animation {
namespace tools {
  class base;
  class animation_timer;
  class animation_path;
  class parabolic_interpolator;
  class linear_interpolator;
  class slerp;
  class animation_factory;
  class separation_modifier;
  class debug_animation_analyser;
  class reference_frame;
}
namespace paths {
  class nurbs;
  class spline;
  class parameteric_animator;
  class bezier_animator;
  class object_animator;
  class colour_animator;
  class colour_animator;
  class vertex_animator;
}
namespace surfaces {
  class bezier_patch_deformer;
  class height_map_deformer;
  class edge_deformer;
  class procedural_texture_animator;
  class discrete_volumetric_modifier;
}
namespace particles {
  class particle_explosion;
  class blingulator;
  class morning_star;
  class rivaller;
  class asteroid;
}
```

Modules

```
// speech.cppm
export module speech;
export const char* get_phrase() {
    return "Hello, world!";
}
// main.cpp
import speech;
import <iostream>;
int main() {
    std::cout << get_phrase() << '\n';
}
```

Modules will solve everything!!!




NO

Start SOMEWHERE

Abstraction resolution

Start SOMEWHERE

Abstraction resolution

Start thinking at ten

Start SOMEWHERE

Abstraction resolution

Start thinking at ten

Divide

Start SOMEWHERE

Abstraction resolution

Start thinking at ten

Divide

Gather

Start SOMEWHERE

Abstraction resolution

Start thinking at ten

Divide

Gather

Abstraction by the rule of ten