Engine 5

ВВЕДЕНИЕ

Engine 5.2.4 – игровой движок, предназначенный для создания двумерных игр. Сам движок ориентирован именно на обработку графики.

Зачем нужен Engine 5.2.4?

Игровой движок предназначен для простых и средней сложности игр. Например, подойдет, для создания игр-стрелялок или платформеров.

ЧТО ПРЕДОСТАВЛЯЕТ ДАННЫЙ ДВИЖОК?

При использовании данного движка в распоряжении пользователя оказываются средства обработки растровой и векторной графики. Парадигмы программирования, используемые при написании кода на данном движке:

- ООП (объектно-ориентированное программирование)
- СОП (событийно-ориентированное программирование)
- ФП (функциональное программирование)

СРЕДА РАЗРАБОТКИ

Средой разработки данной версии движка была PascalABC.Net 3.3.5 (сборка 1662).

АКТУАЛЬНОСТЬ

Проект можно считать уникальным потому, что предоставляет возможность юным программистам пользоваться функционалом языка Pascal в сфере графики. Также за счет того, что диалект языка PascalABC.Net является гибридом C# и старого доброго Pascal, не должно возникнуть проблем в дальнейшем перейти на более современные языки, такие как C#.

ЦЕЛЕВАЯ АУДИТОРИЯ

Данный проект в первую очередь планировался как проект для развития навыков программирования в области написания движков. В последствии этот движок может служить началом для обучения людей в области написания игр в среде PascalABC.NET.

СОСТАВ И РОЛИ В КОМАНДЕ

- Исаков Александр – руководитель и организатор проекта.
  - Тестирование кода
- Волобуев Илья – ведущий разработчик и тестировщик движка.
  - Написание кода графического движка
  - Стандартизация кода в соответствии с стандартом оформления кода C#
  - Тестирование кода
- Жигулин Алексей – разработчик и тестировщик демо-версии игры.
  - Тестирование кода

ПЛАНИРОВАЛОСЬ

- Написать движок.
- Создать игру на базе движка.

РЕЗУЛЬТАТ РАБОТЫ

К данному моменту первую и главную часть проекта можно считать законченной – игровой движок. Демонстрация возможностей движка в разработке.

Далее можете наблюдать скрины кода.

ОБЪЕДИНЕНИЕ ИНТЕРПОЛЯТОРОВ

```
uses Interpolators, Core;
const
  Subdivisions = 50;

var
  InterpolatorUnion: TInterpolatorsUnion;

begin
  InterpolatorUnion := new TInterpolatorsUnion(new TInterpolator(x -> Sin(x), 0, 4 * Pi, Subdivisions),
                                               new TInterpolator(x -> x, 0, 10, Subdivisions),
                                               new TInterpolator(x -> Cos(x), 0, Pi, Subdivisions));
end.
```

ПРИМЕНЕНИЕ ВИЗУАЛИЗАТОРА ИНТЕРПОЛЯТОРА

```
uses Events, Structures, Graphics, Interpolators, VectorPrimitives, Core;
const
  Subdivisions = 50;

var
  InterpolatorVisualizer: TPathVisualizer;

procedure DrawCurve(sender: object; e: TMouseEventArgs);
begin
  InterpolatorVisualizer.Interpolator.Second := new TPoint(e.X, e.Y);
end;

begin
  var (Xinterpolator, Yinterpolator) := (new TInterpolator(x -> x, 0, 100, Subdivisions),
                                         new TInterpolator(y -> Sin(y), 0, 2 * Pi, Subdivisions));
  Xinterpolator.HighY := 100;
  Yinterpolator.LowY := 0.5;
  InterpolatorVisualizer := new TPathVisualizer(new TPoint(100, 100), new TSizeF(300, 300));
  InterpolatorVisualizer.Interpolator := new TPathInterpolator(new TPoint(100, 100),
                                                               new TPoint(300, 300),
                                                               Xinterpolator,
                                                               Yinterpolator);


  Scene.Add(InterpolatorVisualizer);
  TCore.OnMouseMove += DrawCurve;
end.
```

ИЗМЕНЕНИЕ ЦВЕТОВ КЛЕТОК ДОСКИ

```
uses Events, Styles, Graphics, Structures, VectorPrimitives, Core;

var
  Board: TBoard;
  Colors := Arr(new TStyle(clRed), new TStyle(clYellow));

procedure ChangeColors(sender: object; e: TMouseEventArgs);
begin
  Swap(Colors[0], Colors[1]);
end;

begin
  Board := new TBoard(new TPoint(10, 10), new TSizeF(200, 200));
  Board.Styles := Colors;
  Board.Rule := (i, j) -> (i + j) mod 2;

  Scene.Add(Board);

  TCore.OnMouseDown += ChangeColors;
end.
```

БРОУНОВСКОЕ ДВИЖЕНИЕ

```
uses Events, Structures, Graphics, VectorPrimitives, RandomObjects, Core;

var
  Rects: List<TRectangle>;

procedure BrownMoving();
begin
  foreach var rectangle in Rects do
    rectangle.MoveOn(Random(3) - 1, Random(3) - 1);
end;

begin
  Rects := new List<TRectangle>();
  for var i := 0 to 9 do
  begin
    var rectangle := new TRectangle(RandomPoint(), RandomSizeF(200, 250));
    rectangle.FillColor := GetRandomColor();
    Rects.Add(rectangle);
    Scene.Add(Rects.Last());
  end;

  TCore.OnPredraw += BrownMoving;
end.
```

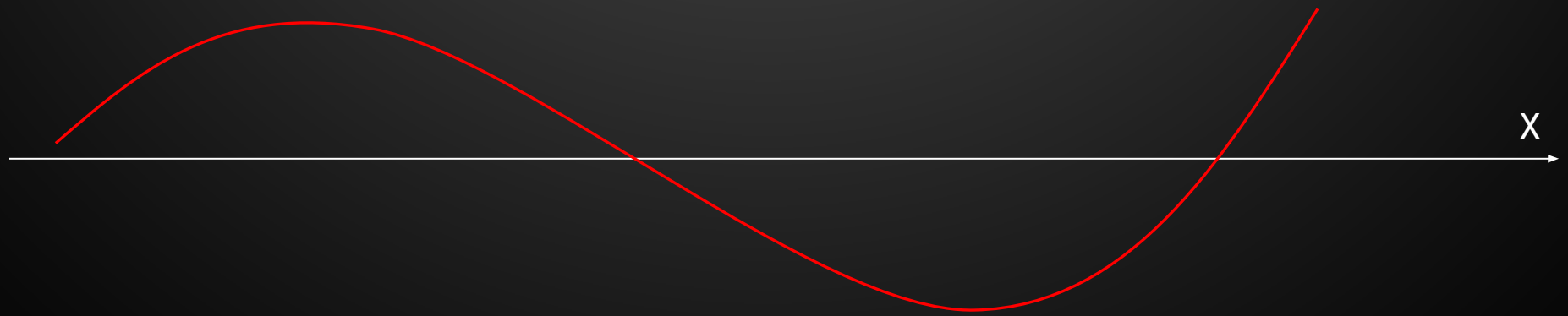
КЛАССЫ ДВИЖКА

В движке имеется множество классов и структур, которые можно разбить по категориям:

1. Классы для интерполирования функций
2. Классы для стилизации объектов
3. Векторные примитивы
4. Игровые классы
5. Классы окружения

КЛАССЫ ДЛЯ ИНТЕРПОЛИРОВАНИЯ ФУНКЦИЙ

С помощью классов интерполирования функций возможно управлять тем, какие Вы хотите получать значения f(x) в зависимости от значения x.



КЛАССЫ ДЛЯ ИНТЕРПОЛИРОВАНИЯ ФУНКЦИЙ

После того как Вы определили функцию, Вам обязательно надо указать пределы интерполирования. Если Вы установите нижнюю границу выше верхней, то они сразу поменяются местами. Также, следует заметить, что предоставляются такие возможности управления интерполяцией функции как:

- Инвертирование значений функции относительно среднего значения на промежутке интерполирования

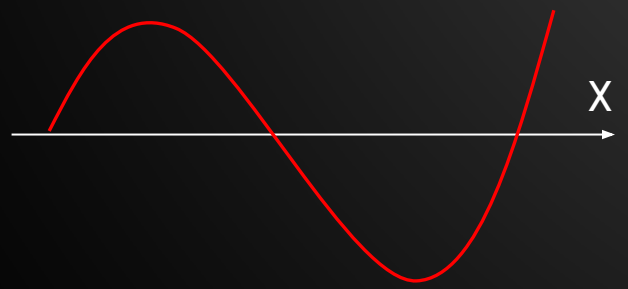
- Нормализация функции (приведение к диапазону [0..1])

- Изменение масштаба функции по оси Y

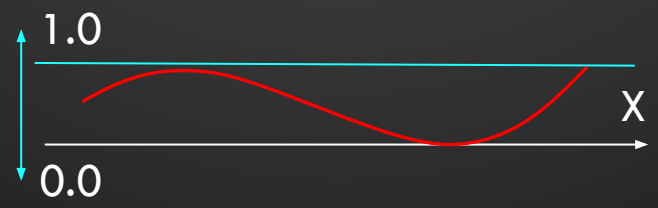
КЛАССЫ ДЛЯ ИНТЕРПОЛИРОВАНИЯ ФУНКЦИЙ

Рассмотрим графики:

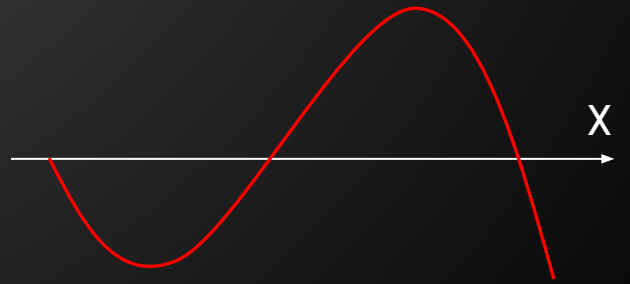
Изначально:



Нормализированный:

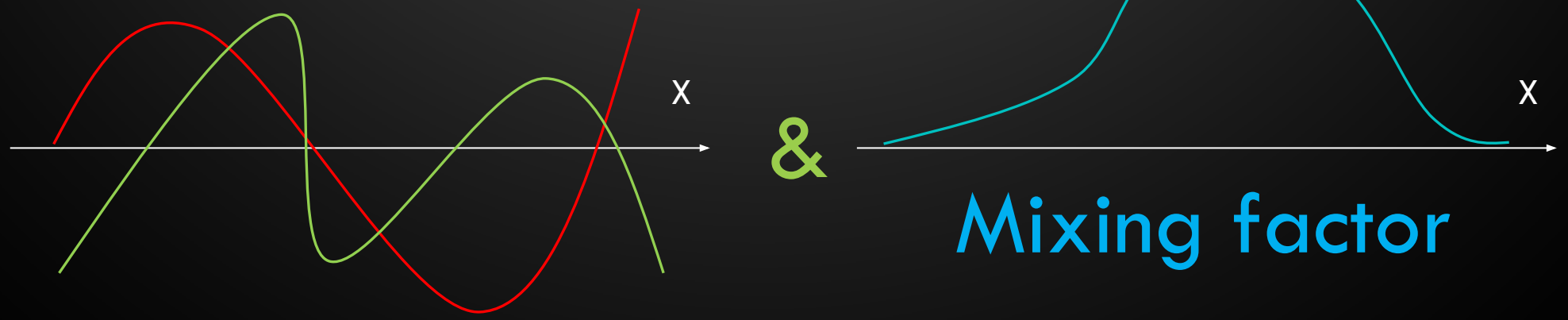


Инвертированный:



КЛАССЫ ДЛЯ ИНТЕРПОЛИРОВАНИЯ ФУНКЦИЙ

Объединение интерполяторов – это класс, который на основе двух интерполяторов, указывающих функции интерполяции, и третьего, указывающего силу смешивания функции первого интерполятора с функцией второго или наоборот предоставляет последовательность f(x).



КЛАССЫ СТИЛИЗАЦИИ ОБЪЕКТОВ

Классы стилизации объектов используются преимущественно при изображении объектов класса TBoard. Так, можно указать массив стилей, используемых доской, а также правило выбора стиля, которое ставит в соответствие двум индексам i и j некоторый индекс массива стилей, который будет применён к клетке с координатами i и j. При попытке отображения доски без установки правила выбора стилей будет выброшено исключение.

(i, j) => index

ВЕКТОРНЫЕ ОБЪЕКТЫ

Существуют классы векторных объектов для таких фигур как:

Отрезок 

Прямоугольник 

Прямоугольник с текстом внутри 

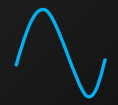
Эллипс 

Эллипс с текстом внутри 

Текст 

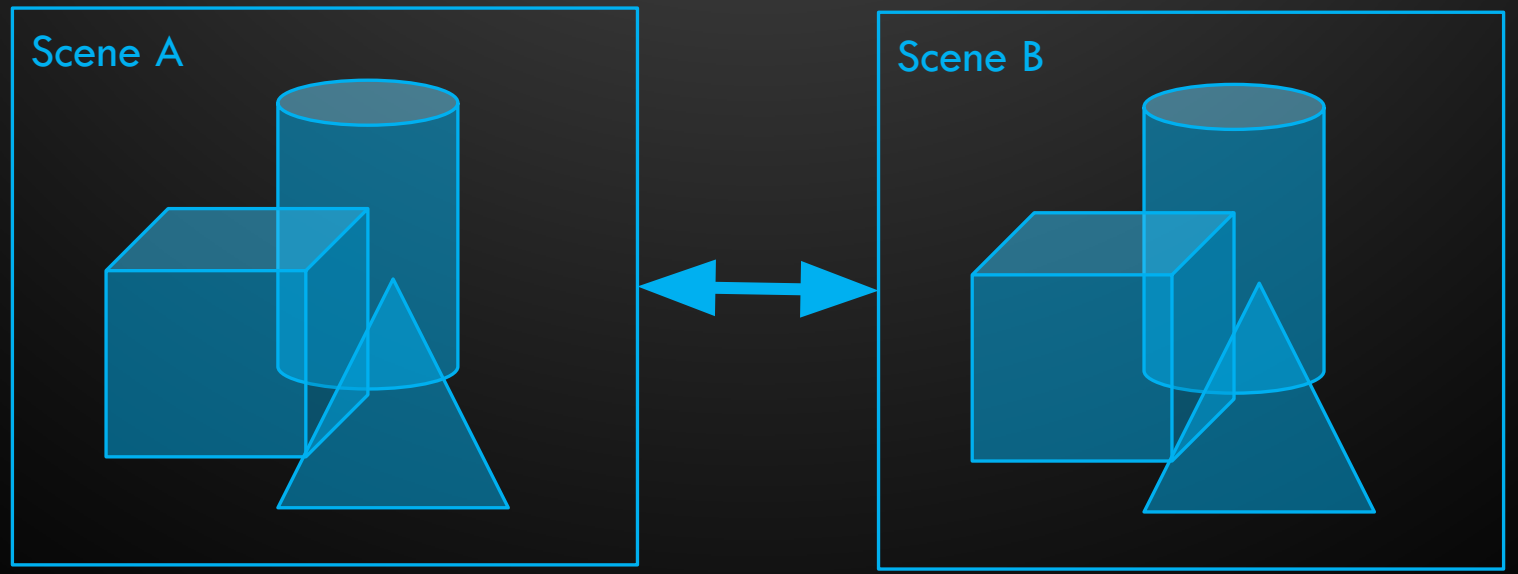
Изображение 

Доска 

Визуализатор интерполятора 

КЛАССЫ ОКРУЖЕНИЯ

Игровой движок поддерживает концепцию так называемых игровых сцен. Их можно воспринимать как отдельные игровые пространства, уровни. При переходе на другой из уровней состояние предыдущего сохраняется.



ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

Модули движка: VectorPrimitives, TextWriter, Styles, Structures, RandomObjects, Players, Main, Interpolators, Graphics (от Engine 4), FP, ExtensionMethods, Events, Core.

Классы движка:

- VectorPrimitives: TOrigin, TBox, TRenderable, TLine, TFillable, TRectangle, TEllipse, TText, TRectangleText, TEllipseText, TImage, TBoard, TPolygon, TPathVisualizer, TDesignes
- TextWriter: TWriter
- Styles: TStyle
- Structures: TSize, TSizeF, TPoint, TRelativePoint, TSizeConverter, TPointConverter
- Players: TPlayer
- Interpolators: TInterpolator, TInterpolatorsUnion, TPathInterpolator
- Events: TEventArgs, TMouseEventArgs, TKeyboardEventArgs
- Core: TScene, TCore

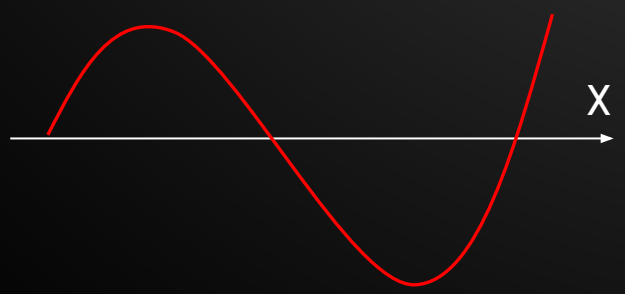
Интерфейсы движка: ICloneableAs<T>

ТЕХНИЧЕСКИЕ ПРОБЛЕМЫ

Одна из причин почему дальнейшее развитие данной версии движка прекратилось – баги среды PascalABC.Net. Данная причина послужила веским аргументов для перехода в другую среду разработки для продолжения написания движка. А точнее – разработки его следующей версии.

ДАЛЬНЕЙШИЕ ПЛАНЫ

Дальнейшее развитие движка заключается в развитии средств для работы с анимацией, системой частиц и эффектами. Некоторые из уже существующих функций движка планируется устранить в следующих версиях, заменив более гибкими. Так, например, планируется заменить класс TUnionInterpolator модификаторами для интерполяторов. Также, планируется добавить средства сохранения и загрузки игровых объектов, возможность отмены каких-либо действий в игре, работа с базами данных.



&