Изготвили:

Цветина Георгиева, СИ, 2 курс, 61696

Симона Янакиева, СИ, 2 курс, 61731

Тодор Желев, СИ, 2 курс, 61738

Графичен редактор STS

Курсова работа по Приложения на математиката за моделиране на реални процеси

Съдържание

[1 Постановка на задачата 3](#_Toc423818137)

[1.1 Различни начини за задаване на криви 3](#_Toc423818138)

[1.2 Рисуване на стандартни равнинни фигури 3](#_Toc423818139)

[1.3 Задаване на ефекти върху изображение 3](#_Toc423818140)

[1.3.1 Зареждане на изображение 3](#_Toc423818141)

[1.3.2 Селектиране на част от изображение 3](#_Toc423818142)

[1.3.3 Ротация 3](#_Toc423818143)

[1.4 Ограничения 3](#_Toc423818144)

[2 Математически основи на реализираните алгоритми 4](#_Toc423818145)

[2.1 Различни начини за задаване на криви 4](#_Toc423818146)

[2.1.1 Криви на Безие 4](#_Toc423818147)

[2.1.2 Сплайн криви 8](#_Toc423818148)

[2.2 Рисуване на стандартни равнинни фигури 11](#_Toc423818149)

[2.2.1 Окръжност 11](#_Toc423818150)

[2.2.2 Елипса 11](#_Toc423818151)

[2.2.3 Триъгълник 12](#_Toc423818152)

[2.2.4 Правоъгълник 12](#_Toc423818153)

[2.2.5 Квадрат 12](#_Toc423818154)

[2.2.6 Линия 13](#_Toc423818155)

[2.3 Задаване на ефекти върху изображение 15](#_Toc423818156)

[2.3.1 Ротация 15](#_Toc423818157)

[3 Описание на крайния продукт 16](#_Toc423818158)

[3.1 Form 16](#_Toc423818159)

[3.1.1 Form UI 16](#_Toc423818160)

[3.1.2 Events 16](#_Toc423818161)

[3.1.3 Graphics 16](#_Toc423818162)

[3.1.4 InputOptions 17](#_Toc423818163)

[3.2 Renderer 17](#_Toc423818164)

[3.2.1 Слоеве 17](#_Toc423818165)

[3.2.2 Метод Render 17](#_Toc423818166)

[3.2.3 Метод RenderLayer 17](#_Toc423818167)

[3.2.4 Метод ClearGraphics 17](#_Toc423818168)

[3.2.5 Метод DrawImage 17](#_Toc423818169)

[3.2.6 Метод DrawPoint 17](#_Toc423818170)

[3.3 Drawables 18](#_Toc423818171)

[3.3.1 Описание на интерфейса 18](#_Toc423818172)

[3.3.2 Point 18](#_Toc423818173)

[3.3.3 Line 18](#_Toc423818174)

[3.3.4 Circle 18](#_Toc423818175)

[3.3.5 B-Spline Function 18](#_Toc423818176)

[3.3.6 Bezier Function 18](#_Toc423818177)

[3.3.7 Rectangle 18](#_Toc423818178)

[3.3.8 Triangle 19](#_Toc423818179)

[3.3.9 Square 19](#_Toc423818180)

[3.3.10 Ellipse 19](#_Toc423818181)

[3.4 Tools 19](#_Toc423818182)

[3.4.1 Eraser 19](#_Toc423818183)

[3.4.2 Selector 19](#_Toc423818184)

[3.4.3 Rotate 19](#_Toc423818185)

[4 Източници 19](#_Toc423818186)

# Постановка на задачата

Да се направи приложение тип графичен редактор (подобно на Paint), което да поддържа следните функционалности:

## Различни начини за задаване на криви

По няколко дадени контролни точки да се построи крива.

## Рисуване на стандартни равнинни фигури

По дадени 2 точки да се чертае кръг, триъгълник, квадрат, правоъгълник, елипса в зависимост от желанието на потребителя. Точките да се задават с движение на мишката.

## Задаване на ефекти върху изображение

### Зареждане на изображение

Потребителят трябва да може да избере изображение, което да се визуализира в приложението.

### Селектиране на част от изображение

Потребителят трябва да може да избере част от полето.

### Ротация

Селектираната част от полето да се завърта под избран ъгъл.

## Ограничения

Да се не се използват готови функции и обекти за реализацията на приложението.

Да се имплементират всички използвани алгоритми собственоръчно.

За чертаенето на всички обекти да се използва единствено вградена функция за чертаене на един пиксел със зададен цвят на зададено масто.

# Математически основи на реализираните алгоритми

## Различни начини за задаване на криви

### Криви на Безие

Крива на Безие е параметрична крива, често използвана в компютърната графика.

#### История[[1]](#footnote-1)

Математическата база за кривите на Безие – полиномът на Бернщайн – е познат още от 1912, но приложението му в графиката е разбрано едва половин век по-късно. Кривите на Безие били широко публицирани в 1962 от френския инженер Пиер Безие, който ги използвал при дизайна на автомобилни шасита за Рено. Изследването на тези криви обаче първи разработил математикът Пол дьо Кастелажо през 1959, разработвайки алгоритъма, който носи неговото име, устойчив метод за построяване на криви на Безие, в Ситроен, друга френска автомобилна компания.

#### Приложение[[2]](#footnote-2)

##### Компютърна графика

Кривите на Безие са широко използване в компютърната графика за моделиране на гладки криви. Тъй като кривата изцяло се намира в полуравнината на контролните си точки, те могат графично да се изобразят и да се използват за интуитивна промяна на кривата.

Най-простият метод за изрисуване на една крива на Безие е да се оцени в много близко намиращи се точки и да се изрисува последователността от прави, които ги свързват. Това обаче не гарантира, че изходното изобразяване ще изглежда достатъчно гладко, защото точките могат да са прекалено отдалечени. И обратното – може да се генерират прекалено много точки в участъци, където кривата е близка до линейна. Често използван метод е рекурсивно подразделяне, в което контролните точки на кривата се проверяват, за да се види дали кривата наподобява права (използва се определена точност). Ако не, кривата се разделя на два сегмента за 0 ≤ t ≤ 0.5 и 0.5 ≤ t ≤ 1 и същата процедура се прилага рекурсивно за всяка част.

##### Анимации

В приложения за анимация като Adobe Flash и Synfig, криви на Безие се използват за описание например на движение. Потребителят очертава искания път с криви на Безие и приложението създава необходимите кадри, така че обектът да се движи по пътя.

##### Шрифтове

TrueType шрифтовете използват по части квадратни криви на Безие. Модерните системи за изображения като PostScript, Asymptote, Metafont, и SVG използват по части криви на Безие от 3-та степен за рисуване на закръглени форми. OpenType шрифтовете използват и двата вида в зависимост от шрифта.

Вътрешното изчертаване на всички криви на Безие в шрифтовата и векторната графика ги разделя рекурсивно до момент, в който кривата е достатъчно плоска, за да се изчертае като серия от линейни или кръгови сегменти. Точните алгоритми за разделянето зависят от имплементацията, докато критерият за плоскост трябва да се използва, за да се достигне необходимата прецизност и да се избегнат немонотонни локални промени на извивката. Диаграмите в Microsoft Excel също използват този алгоритъм при функционалността за “Гладка крива”

#### Обща дефиниция[[3]](#footnote-3)

За задаване на крива на Безие от степен n са необходими:

* n + 1 на брой контролни точки **P**0, **P**1, ..., **P***n*, където *n* ∈ N. При последователното си свързване тези точки образуват контролния полигон на кривата. Отсечките **P**i **P**i+1 (*i* = 0, 1, ..., *n* − 1), свързващи две последователни контролни точки, определят т. нар. контролни рамене.
* Полиномите на Сергей Бернщайн (основни функции на Безие), означени с **B**n,i(u) . Това са полиноми на една реална променлива u, определени чрез:
* Тогава кривата на Безие **C**(u) от *n*-та степен,определена от контролните точки **P**0, **P**1, ..., **P***n*, има вида



* Параметърът *u* на кривата се изменя в затворения интервал [0, 1].

#### Свойства[[4]](#footnote-4)

* Кривата започва в **P**0 и завършва в **P***n*.
* Кривата представлява права тогава и само тогава, когато всички контролни точки са колинеарни.
* Началото (краят) на кривата е допирателно до първото (последното) контролно рамо от полигона на Безие.
* Кривата може да се раздели в коя да е точка на две под-криви, или съответно на много под-криви, всяка от които също е крива на Безие.
* Някои криви, които изглеждат прости, например окръжност, не могат да се опишат точно с крива на Безие или по части крива на Безие. Въпреки това крива на Безие от 4 части може да приближава окръжност с максимална грешка на радиуса по-малка от 1/1000, когато всяка вътрешна контролна точка е разстояние \textstyle\frac{4\left(\sqrt {2}-1\right)}{3}хоризонтално или вертикално от външна контролна точка на единична окръжност. По-общо казано, крива на Безие от n части може да приближава окръжност, когато всяка вътрешна контролна точка е разстояние \textstyle\frac{4}{3}\tan(t/4) от външна контролна точка на единична окръжност, където t е 360/*n* градуса, и *n* > 2.
* Всяка квадратна крива на Безие е също и кубична крива на Безие или по-общо, всяка крива на Безие от степен n е също крива на Безие от степен m за всяко m > n. По-точно крива на Безие от степен n с контролни точки **P**0,..., **P***n* е еквивалентна(включително параметризация) на крива от степен n+1 с контролни точки **P'**0, …, **P'***n*+ 1, където \mathbf P'_k=\tfrac{k}{n+1}\mathbf P_{k-1}+\left(1-\tfrac{k}{n+1}\right)\mathbf P_k.
* Кривата не изменя формата си при промяна на координатната система (афинна инвариантност)[[5]](#footnote-5).
* Кривите на Безие притежават свойството променливо намаляване (variation diminishing property)[[6]](#footnote-6). Никоя права не пресича равнинна крива на Безие повече пъти отколкото пресича контролния й полигон.
* Поради свойството "разделяне на единицата" и неотрицателността на полиномите на Бернщайн, всяка крива на Безие лежи изцяло в изпъкналата обвивка на контролните си точки. Това е най-малкият изпълкнал многоъгълник (т.е. с ъгли < 180◦ ), който съдържа във вътрешността си или върху контурите си всички контролни точки. Изпъкналата обвивка на две точки е отсечката, която ги съединява. Изпъкналата обвивка на три точки, нележащи върху една права, е триъгълникът с върхове тези точки. На четири точки – изпъкналият четириъгълник, образуван от тях и т.н.
* Няма локален контрол в крива на Безие от ред n. Това означава, че всяка промяна на контролна точка изисква повторно пресмятане и така има влияние върху цялата крива. Въпреки това колкото по-далеч е точката от кривата спрямо променената контролна точка, толкова по-малко е изменението.

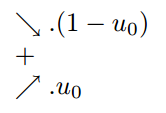
#### Построяване на криви на Безие чрез алгоритъма на Дьо Кастелжо[[7]](#footnote-7)

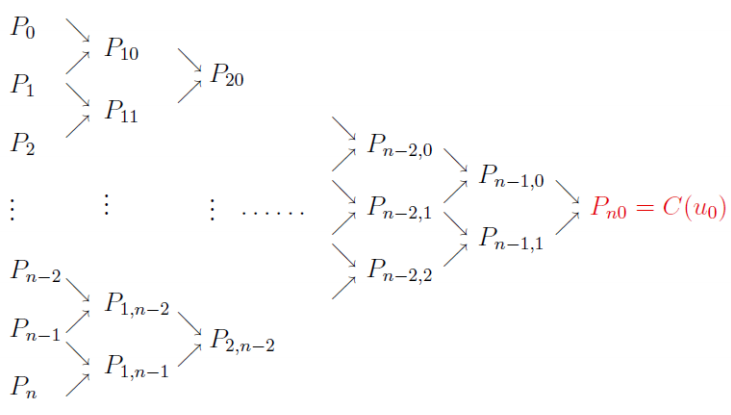
Нека **C**(*u*)*, u* ∈ [0, 1], е крива на Безие от *n*-та степен, определена от контролните точки **P**0, **P**1, ..., **P***n*. За да намерим точка от кривата, съответна на стойност на параметъра *u* = *u0*, т. е. точката C(*u0*), използваме алгоритъма на дьо Кастелжо. При работата с този алгоритъм, вместо да бъде използвана параметричната форма на кривата, се работи само с нейния контролен полигон.

Алгоритъмът се базира на следната идея. Ако AB е отсечка и търсим точка M върху нея, разделяща я в отношение AM : MB = *u0* : 1 − *u0*, координатите на тази точка се изразяват като следната линейна комбинация от координатите на двата края на отсечката M = (1 − *u0*)A + *u0*B.

Намирането на C(*u0*) чрез алгоритъма на дьо Кастелжо се състои в следното:

Подреждаме контролните точки на кривата вертикално една над друга, започвайки от **P**0 и стигайки до **P**n, както е показано в най-лявата колона на схемата.

Аритметичните действия в схемата са символично изобразени по следния начин:



В първия етап на алгоритъма намираме точка **P**1i върху всяко рамо (отсечка) **P**i **P**i+1 (i = 0, 1, . . . , n−1) на контролния полигон, която го дели в отношение *u0* : (1 − *u0*). Тази точка се получава по формулата **P**1i = (1 − *u0*) **P**1+ *u0***P**i+1. Използваме горната схема.

По този начин получаваме точките **P**10, **P**11, . . . , **P**1,n-1 от следващата колона (втората отляво надясно). Процедурата се повтаря с получаване на точките **P**2i (i = 0, 1, . . . , n−2) от третата колона, които лежат съответно върху отсечките **P**1i **P**1,i+1. След това се пресмятат координатите на точките от четвъртата колона и така нататък до получаването на последната точка **P**n0. Съгласно метода, тази точка съвпада с C(u0). Построявайки начупените линии по всяка колона на схемата, на- мираме т. нар. мрежа на дьо Кастелжо, която постепенно приб- лижава кривата, докато достигне точката C(u0).

На следващите схеми може да видим как изглежда графично приложеният алгоритъм.[[8]](#footnote-8)

##### Линейни криви



Тук контролните точки са две – **P**0и **P**1.

За дадено *u0* определяме отношението *u0* : 1 - *u0*.

Отсечката **P**0**P**1 делим в същото отношение и получената точка е търсената.

##### Квадратични криви



Тук контролните точки са три - **P**0**, P**1и **P**2 . Отново първо определяме отношението *u0* : 1 - *u0*.

Контролните рамена **P**0**P**1 и **P**1**P**2 делим в същото отношение и получените точки означаваме с **Q**0 и **Q**1.

Отсечката **Q**0**Q**1 делим в същото отношение и получената точка е търсената.

##### Криви от по-висок ред



Нека контролните точки са n - **P**0**, P**1**,..,,P**n . Отново първо определяме отношението *u0* : 1 - *u0*.

Контролните рамена **P**0**P**1,…, **P**n-1**P**n делим в същото отношение и получените точки означаваме с **Q**0,..., **Q**n-1.

На втора стъпка, отсечките **Q**0**Q**1, ..., **Q**n-2**Q**n-1 делим в същото отношение и получените точки означаване с **R**0,..., **R**n-2.

Продължаваме алгоритъма за отсечките **R**0**R**1, ..., **R**n-3**R**n-2 и т.н. докато не получим само една точка, която всъщност е търсената точка от кривата на Безие.

### Сплайн криви[[9]](#footnote-9)

Точността на приближение на дадена функция  в крайния интервал  зависи съществено от дължината на интервала и степента на алгебричния полином. Тъй като компютърните пресмятания с полиноми от висока степен водят до известни проблеми, то желателно е да се използват полиномиот невисоки степен. Тогава единственият шанс за увеличаване на точността на приближение идва от работа върху малки нтервали. Ако интервалът  е голям, той се разделя на малки подинтервали , и  се приближава в  с алгебричен полином  от някаква ниска степен . По този начин получаваме приближението

 **за** .

Функцията  представлява една на части полиномиална крива, която приближава графиката на  с определена точност. В общия случай  е прекъсната в точките . Ако  описва гладък процес, то желателно е и приближаващата функция да бъде гладка. За да се постигне това, на полиномиалните части се налага допълнително условие да се свързват гладко, т.е. производните на  и  до определен ред да съвпадат в точката на свързване . В резултат се получава една гладка крива, която приближава добре. Такива криви се наричат сплайн-функции. Наименованието идва от един стар уред за чертаене на гладки криви през зададени точки, наречени „сплайн.“

Това е един от начаните да се обясни появата на сплайн-функциите в математиката – като апарат, който е роден от нуждите на практиката. Интересни свойства на сплайн-функциите и дълбоките им връзки с други направления в математиката обаче показват, че появата на сплайн – функциите е обусловена от вътрешната логика на развитие на самата математика. Теорията на сплайн-функциите е една от най-бурно развиващите се области на анализа в последните 30 години.

Определение:Фунцкията  за , се нарича сплайн-функция от степен  с възли , ако:

1.  е полином от степен  най-много във всеки подинтервал

, , 

1.  са непрекъснати функции в .

#### -сплайни

Вече показахме, че всеки сплайн от степен  с възли може да бъде представен като линейна комбинация на полином  от и отсечените степенни функции

.

Определение:Разделената разлика на отсечената степенна функция  по отношение на  в точките  се нарича -сплайн от степен с възли .

Теорема . При всяко имаме:

1. за всяко  и всяко ,
2. при .

Теорема.Нека са фиксирани точки. Да изберем произволни други  точки  и . Нека, .

-сплайнитеобразуват базис в пространството върху интервала 

И така, всяка сплайн-фунцкия от може да бъде представена по единствен начин във вида

.

Имайки предвид крайния носител на , това е много удобно представне на за работа с компютър, тъй като при фиксирано , сплайнът е всъщност линейна комбинация само на последователни –сплайни, които съдържат  в своя носител. Едно друго предимство на представянето е, че съществува проста схема за пресмятане стойността на  в дадена точка. Тази схема се основава на следната рекурентна връзка.

Основна рекурентна връзка:За всяко е в сила равенството e в сила равенството

.

Да отбележим, че коефициентите пред  и  в рекурентната връзка са положителни при  и тяхната сума е равна на 1. Следователно формулата представя като изпъкнала комбинация на  и .



## Рисуване на стандартни равнинни фигури

Всеки нов създаден клас за рисуване на равнинни фигури са public и се достъпват от всяко място в приложението.

### Окръжност

В приложението имаме създаден нов клас за рисуване на окръжност – class Circle. Създава се списък от точки.

#### Рисуване на окръжност



Взимаме координатите на центъра и радиуса на окръжността чрез мишката. Всяка точка има свои полярни координати съставени от радиуса на окръжността и стойност на ъгъл, вариращ от 0 до 360 градуса. Самото чертаене става чрез свързване на първата точка до затваряне на окръжността чрез линия.

### Елипса

Клас за рисуване на елипса – class Ellipse.

#### Рисуване на Елипса



За изчертване на елипса се използва уравнението на елипса

Където е центърът на елипсата, а е главната полуос, b е малката полуос. Първо се изчертава горната част на елипсата, след това долната, като получените точки се свързват последователно с линии.

При mouse down избираме координатите на първата точка Р1, a последната точка P2 се намира при mouse up на мишката. Същестуват два радиуса, наречени малка и главна полуос. Главната полуос е с координати, които се намират като се вземе по абсолютна стойност половината от разликата между Х-координатите на Р1 и Р2. Малката полуос е с координати, които се намират като се вземе по абсолютна стойност половината от разликата между У-координатите на Р1 и Р2. Центърът се намира с координати от половината от сборовете на Х и У координатите на началната и крайната точка.

### Триъгълник

Клас за рисуване на триъгълник – class Triangle.

#### Рисуване на триъгълник



За изчертаването на триъгълник ни трябва начална точка P1 с координати (a,b) и крайна точка P2 с координати (c,d). Те са координати на невидим правоъгълник, чрез който избираме мястото, където да се начертае фигурата. На картинката се вижда чрез координатите на точка Р3 са Х-координата на точка Р1 и У-координата на точка Р2. Между Р3 и Р2 се рисува линия от точки. Върхът на равнобедрения триъглъник е намерен с чрез половината от сбора Х-координатите на Р1 и Р2. У-координата е взета от началната точка Р1. Свързват се трите точки Р3, Р2 и Р4 чрез последователни линии.

### Правоъгълник

Клас за рисуване на правоъгълник – class Rectangle.

#### Рисуване на правоъгълник



За изчертаването на правоъгълник са нужни координати на първата и последната точка. Р3 се получава от Х-координатата на Р1 и У-координата на Р2. Р4 се получава от Х-координатата на Р2 и У-координата на Р1. Рисуването става чрез свързване на точките чрез линии.

### Квадрат

Клас за рисуване на квадрат– class Square.

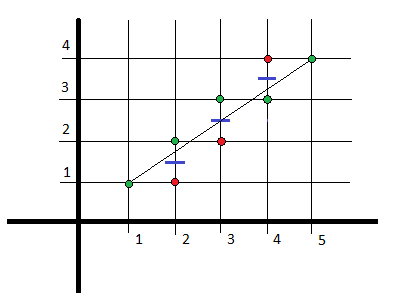
#### Рисуване на квадрат



За изчертаването на квадрата са нужни координати на първата точка Р1. Последната точка има своята особеност. За да не се получи отново правоъгълник при функционалността mouseUр, която избира последната точка, се изчислява абсолютната стойност от разликата на крайната и началната точка, измерена по У-координата, наречена в нашата програма diff. По този начин крайната точка има вида – Х-координата на Р1 + diff и своята У-координата. Р3 има стойност Х-координата на Р1 и У-координата със стойност У-координата на Р1+ diff. Р4 има стойност X-координата със стойност X-координата на Р1+ diff и У-координата на Р1. Рисуването става чрез свързване на точките чрез линии.

### Линия

Генерирането на точки за изчертаването на линия се осъществява чрез алгоритъма на Брезенхам. На следната картинка е даден пример за избиране на точките за рисуване на линия между точките (1,1) и (5,4)



В алгоритъма са основни 4 променливи – наклон, праг, отместване и текущ . В нашия пример наклонът на линията ще бъде . Първоначално прагът е , а отместването е . Текущият е координатата на началната точка.

Идеята на алгоритъма е да итерираме през координатите от началната точка до крайната, като за всеки проверяваме дали текущото отместване е по-голямо или равно на текущия праг и ако е така, увеличаваме текущия с 1 и увеличаваме прага с 1. Също така за всеки добавяме точката с координати и текущия и увеличаваме отместването с наклона на линията.

На картинката със синя линия са изобразени текущите прагове за всеки . Зелените точки са точките, които ще бъдат избрани за рисуване, а червените –не.

Сега ще покажа как работи алгоритъма за всяко от примера. означава, че точката ще бъде добавена за рисуване.

В този момент алгоритъмът приключва тъй като сме достигнали x координатата на крайната точка. Окончателно за рисуване сме добавили точките

В програмата също така са покрити случайте, когато началният е по-голям от крайния (когато рисуваме линия отдясно наляво) и когато наклонът на линията е отрицателно число. Осъществява се като разменяме нужните координати и след това прилагаме алгоритъмът описан по-горе.

## Задаване на ефекти върху изображение

В проекта е реализирано завъртане на област от изображение.

### Ротация

За да извършим ротация първо селектираме област от точки с даден център (𝑐𝑥,𝑐𝑦) чрез инструмента за селектиране.

Тъй като стандартната матрица за завъртане завърта само около началото на координатната система, а не около произволен център трябва да приложим и транслиране.

Точките от селектираната област се транслират до началото на координатната система (0,0), завъртат се на 90 градуса (в програмата е заложено да завърта по default на 90 градуса) по часовниковата стрелка и след това се транслират обратно до началния център (𝑐𝑥,𝑐𝑦).

За да получим координатите на новата точка правим следното умножение, като използваме матрица, която комбинира ротация и транслация

Така x координата на новата точка ще бъде:

а y координатата ще е:

# Описание на крайния продукт

## Form

### Form UI

– форма, в която са всички бутони и място, където извършваме всички действия. Тя има своите ограничения за рисуване - размер 900х700. В програмния код се достига чрез името STS.

#### ToolBox

- от кутията за инструментите създаваме основните бутони, които са ни необходими за създаване на приложението. Всеки бутон има свои функции, които може да променяме.

#### ColorBox

- тук се намират цветовете, които може да използваме за нашите фигури, чрез които може да ги рисуваме. Имаме право да избираме на различни цветове, както и да създаваме нови, като може да ги запазваме и в бъдеще да ги използваме. Има създадено текстово име ColorBox с фон и различен font. В програмния код се достига чрез името ColorButton.

#### Brush Size

– избираме различната дебелина на линия, с която желаем да чертаем. Има три различни видове линии, създадени чрез инструмент на ToolBox-a, наречено ContextMenuStrip. Изброени са три вида - малка, средна, голяма. Достъпват се чрез натискане с ляв или десен бутон на мишката върху бутона за линии. В програмния код се достига чрез името brush.

#### File

– падащо меню, създадено чрез инструмент от ToolBoх, наречено MenuStrip. В него има създадени функции New, Open, Save, Exit. При избрана команда New формата се изчиства и отновно може да се рисува. При избрана команда Open се отваря запазен документ, който сме запазили при команда Save. При команда Exit излизаме от цялото приложение.

### Events

В приложението се използват 3 основни събития – MouseDown, MouseMove и MouseUp.

#### MouseDown

При натискане на левия бутон на мишката, командата се изпълнява и се предава на избрания Drawable обект (от InputOptions).

#### MouseMove

Осъществява се при движение на мишката, след като е натиснат и задържан бутон на мишката.

#### MouseUp

Осъщестява се след като натиснат бутон на мишката е пуснат. При пускане на левия бутон на мишката се подава известие към Renderer, който изчертава подадената информация от CurrentFigure.

### Graphics

Това е системен обект, вграден в WFA, който се използва за чертаене върху полето.

### InputOptions

InputOptions държи информация за цвета, дебелината на линията, обекта, който трябва да се начертае, и инструмента, който се ползва.

## Renderer

Renderer отговаря за рисуването върху полето.

### Слоеве

#### Класът Layer

Този клас играе ролята на слой върху рисувателното поле. Той се характеризира с начална и крайна точка (горен ляв и долен десен ъгъл на правоъгълник от полето). Всеки Layer съдържа матрица от цветове (colorMatrix). На всеки цвят с индекс i, j в тази матрица отговаря пиксел с индекс StartX + i, StartY + j от рисувателното поле. Имплементирани са методите Get и Set, които по зададени координати от полето съответно връщат цвета на пиксела или го заменят с даден цвят. Методът SetMultiple копира цял един слой върху друг.

#### Видове слоеве в Renderer

В класа Renderer се пазят три слоя – field, pastDrawing и currentDrawing. Field представлява слой, който окончателно е запазен върху рисувателното поле. PastDrawing пази състоянието на рисувателното поле преди последното рисуване, а currentDrawing пази последното рисуване.

### Метод Render

Този метод отговаря за рисуването на Drawable обект. По зададени точки и InputOptions се чертаят точките. Тук са реализирани и слоевете.

### Метод RenderLayer

Този метод отговаря за рисуването на даден Layer обект. Самият обект пази точно къде да се нарисува, както и цветовете, които да се изрисуват. Методът се използва при работата с инструменти като Селектиране и Завъртане.

### Метод ClearGraphics

Рисувателното поле се изчиства. Премахват се всички предишни слоеве. Резултатът е същия като да стартираме приложението наново.

### Метод DrawImage

По зададен обект от тип Bitmap се чертае изображение в началото на рисувателното поле в мащаб 1:1 . Ако изображението е по-голямо от полето, се чертае само частта, която попада в него.

### Метод DrawPoint

По зададени координати на точка от полето и цвят се чертае 1 пиксел с дадения цвят. Използва се вградена функция.

## Drawables

Всеки един елемент за рисуване има своя създадена картина и свое име, чрез което се достига в приложението.

### Описание на интерфейса

#### Метод GetPoints

Връща точките, с които е подходящо да се опише фигурата графично. Обикновено тук е алгоритъма, с който от зададените точки от потребителя се изчисляват тези точки.

#### Събитията mouseDown, mouseUp и mouseMove

Използват се за запаметяване на координатите на мишката, които са необходими за задаване на обекта.

#### Свойството NeedConnectPoints

Определя дали отделните точки, генерирани в метода GetPoints е необходимо да бъдат свързани с линия помежду си, за да изглежда графичното им представяне гладко.

#### Свойството NeedRemovePastLayer

Определя дали да се изчисти полето от предишното нарисувано при натискане на повече точки от потребителя. Използва се при рисуването на криви.

### Point

- рисуваме точка или рисуваме линии чрез продължително натискане на левия бутон на мишката (MouseDown) чрез движение.

### Line

– чертаем линия. Използва се алгоритъма на Брезенхам. Чертаенето се осъщестява като се намерят координатите на първата точка чрез mouse down и се прекратява чертаенето при намиране на координатите на втората точка чрез MouseUp.

### Circle

– чертаем кръг. Чертаенето се осъщестява като се намерят координатите на първата точка чрез mouse down, последната точка се намира при MouseUp на мишката, като това е радиусът на кръга. Радусът се изчислява като корен-квадратен от сбора на двете разлики между координатите на точките, повдигнати на степен.

### B-Spline Function

– Б-сплайн криви. Всяка крива не е длъжна да минава през първата и последната си контролна точка, нито да се допира до първото и последното контролно рамо на контролния си полигон.

### Bezier Function

– крива на Безие. Всяка крива минава през първата и последната си контролна точка, създадени от MouseDown и MouseUp.

### Rectangle

– чертаем правоъгълник. Чертаенето се осъщестява като се намерят координатите на първата точка P1 чрез МouseDown, a последната точка P2 се намира при MouseUp на мишката.

### Triangle

– чертаем триъгълник. Чертаенето се осъщестява като се намерят координатите на първата първата и последната точка.

### Square

– чертаем квадрат. Чертаенето се осъщестява като се намерят координатите на първата първата и последната точка.

### Ellipse

– чертаем елипса. При mouse down избираме координатите на първата точка Р1, a последната точка P2 се намира при mouse up на мишката. Същестуват два радиуса, наречени малка и главна полуос. Главната полуос е с координати, които се намират като се вземе по абсолютна стойност половината от разликата между Х-координатите на Р1 и Р2. Малката полуос е с координати, които се намират като се вземе по абсолютна стойност половината от разликата между У-координатите на Р1 и Р2. Центърът се намира с координати от половината от сборовете на Х и У координатите на началната и крайната точка.

## Tools

### Eraser

– това е гума. При mouse move и mouse down на потребител се запазват точките, през които е минал, като запазените точки се свързват последователно с линии. При mouse up цветът на тези точки и точките, генерирани от линиите, става бял, което симулира изтриване.

### Selector

– селектиране на дадена част, избрана чрез мишката.При mouse down и mouse up се запазват две точки (P1,P2), които дефинират правоъгълна област. Тази правоъгълна област от точки се запазва и се използва в другите инструменти.

### Rotate

–завъртане(ротация) на селектирана част. Селектираната част се върти 90 градуса по обратно на часовниковата стрелка. Функцията се осъщестява върху настискането на самия бутон във WFA.

# Източници

Криви на Безие:

<https://en.wikipedia.org/wiki/B%C3%A9zier_curve> – към 04.07.2015

<http://web.uni-plovdiv.bg/marta/GD/lecture3.pdf> – към 04.07.2015

Сплайн криви:

Борислав Боянов, Лекции по числени методи, изд. Дарба София, 2008

1. Източник - <https://en.wikipedia.org/wiki/B%C3%A9zier_curve> [↑](#footnote-ref-1)
2. Източник - <https://en.wikipedia.org/wiki/B%C3%A9zier_curve#Applications> [↑](#footnote-ref-2)
3. Източник - <http://web.uni-plovdiv.bg/marta/GD/lecture3.pdf> [↑](#footnote-ref-3)
4. Източник - <https://en.wikipedia.org/wiki/B%C3%A9zier_curve> [↑](#footnote-ref-4)
5. Източник - <http://web.uni-plovdiv.bg/marta/GD/lecture3.pdf> [↑](#footnote-ref-5)
6. Източник - <https://en.wikipedia.org/wiki/Variation_diminishing_property> [↑](#footnote-ref-6)
7. Източник - <http://web.uni-plovdiv.bg/marta/GD/lecture3.pdf> [↑](#footnote-ref-7)
8. Източник - <https://en.wikipedia.org/wiki/B%C3%A9zier_curve#Constructing_B.C3.A9zier_curves> [↑](#footnote-ref-8)
9. Източник – Борислав Боянов, Лекции по числени методи, 2008 [↑](#footnote-ref-9)