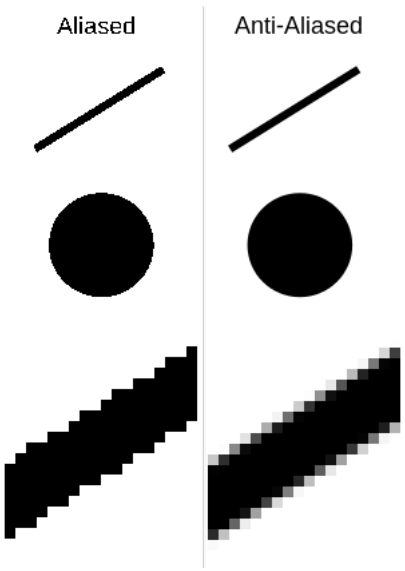
Семинар #2: Библиотека SFML. Событийно-ориентированное программирование. Классные задачи.

- **Фигуры**: В SFML есть несколько классов для работы с простыми фигурами: `sf::CircleShape` (круг или элипс), `sf::RectangleShape` (прямоугольник), `sf::ConvexShape`(фигура сложной формы, задаваемая точками). У этих классов есть общие методы:
  - `setOrigin` - установить центр фигуры (по умолчанию – верхний левый угол)
  - `setPosition`, `getPosition` - задать и получить координаты центра
  - `move` - принимает 2D вектор и передвигает фигуру на этот вектор
  - `setRotation`, `getRotation` - задать и получить угол вращения
  - `rotate` - принимает вещественное число и вращает фигуру на этот угол (в градусах)
  - `setScale`, `getScale` - задать и получить величину масштабирования (2D вектор)
  - `scale` - принимает 2D вектор и растягивает или сжимает фигуру по x и по y соответственно
  - `setFillColor`, `getFillColor` – устанавливает цвет

  В папке `0shapes` приведён пример программы, которая рисует несколько фигур. Используйте методы выше и сделайте так, чтобы круг двигался по прямой, прямоугольник вращался, а фигура сложной формы сжималась по y и растягивалась по x. Подберите скорости этих операций, чтобы они были не слишком быстрыми.

- **Anti-Aliasing**:



Вы могли заметить, что фигуры выглядят не очень красиво - имеют зазубрены. Это связано с тем, что рисования происходит на прямоугольной сетке пикселей и при проведении линий под углом образуются ступеньки. Для борьбы с этим эффектом был придуман специальный метод сглаживания, который называется антиалиасинг. Он уже автоматически реализован во всех библиотеках компьютерной графики. Чтобы установить его в SFML, нужно прописать опцию:

```
sf::ContextSettings settings;
settings.antialiasingLevel = 8;
```

И передать `settings` на вход для конструктора `RenderWindow`.

Протестируйте программу с разными уровнями антиалиасинга.

- **Кадры в секунду:** Сейчас основной цикл программы работает без перерывов и, так как наша программа очень проста, то количество кадров в секунды(fps) может достигать огромных значений - больше 1000 fps. Мониторы не обновляют экран с такой скоростью и человеческий глаз тоже не способен воспринять такую частоту кадров. Поэтому не имеет смысла задавать fps очень высоким. Ограничьте число кадров в секунду 60-ю с помощью метода `setFramerateLimit` класса `RenderWindow`.

- **KeyPressed:** В папке `1key_events` лежит пример программы, которая обрабатывает нажатия клавиш. Измените программу так, чтобы при нажатии на клавишу Enter кружок менял цвет на случайный.

- **KeyReleased:** Измените программу так, чтобы при *отпускании* клавиши пробел прямоугольник менял цвет на случайный (событие `sf::Event::KeyReleased`).

- **isKeyPressed:** – это специальная функция, с помощью которой можно проверить, зажата ли какая-либо клавиша. Она не относится к событиям и её можно вызывать в любой части программы, а не только в цикле обработки событий. При зажатии клавиши эта проверка будет срабатывать каждый раз (в отличии от `KeyPressed`, которая срабатывает в момент нажатия). Сделайте так, чтобы шарик двигался при нажатии на стрелочки (либо на клавиши WASD).

- **MouseButtonPressed:** В папке `2mouse_events` лежит пример программы, которая обрабатывает нажатия и движение мыши. Измените программу так, чтобы при нажатии на правую кнопку мыши, прямоугольник перемещался к положению мыши. Событие должно срабатывать только в момент нажатия, прямоугольник не должен двигаться при зажатии кнопки.

```
if (event.type == sf::Event::MouseButtonPressed)
{
    if (event.mouseButton.button == sf::Mouse::Right)
    {
        std::cout << "the right button was pressed" << std::endl;
        std::cout << "mouse x: " << event.mouseButton.x << std::endl;
        std::cout << "mouse y: " << event.mouseButton.y << std::endl;
    }
}
```

- **isButtonPressed:** – это специальная функция, с помощью которой можно проверить, зажата ли какая-либо кнопка. Она не относится к событиям и её можно вызывать в любой части программы, а не только в цикле обработки событий. При зажатии кнопки эта проверка будет срабатывать каждый кадр (в отличии от `MouseButtonPressed`, которая срабатывает в момент нажатия). Измените программу так, чтобы при зажатии правой кнопки мыши, круг перемещался к положению мыши.

- **MouseMoved:** Событие, которое срабатывает тогда, когда двигается мышь.

```
if (event.type == sf::Event::MouseMoved)
{
    std::cout << "new mouse x: " << event.mouseMove.x << std::endl;
    std::cout << "new mouse y: " << event.mouseMove.y << std::endl;
}
```

  Измените программу так, чтобы прямоуголиник окрашивался в красный цвет тогда и только тогда, когда курсор мыши находится на прямоугольнике. Во всё остальное время прямоуголиник должен быть зелёным.

- **Кнопка:** Создайте "кнопку"из прямоугольника. Логика работы должна быть аналогичной логике работы обычной кнопки в ОС Windows. При нажатии на прямоугольник он должен немного менять цвет. При отпускании мыши, если курсор всё ещё находится на прямоугольнике, должно срабатывать некоторое действие. В качестве действия – пусть круг будет менять цвет на случайный. (в этом задании перемещение круга к курсору при нажатии левой кнопки мыши нужно отключить).

- **Перетаскивание:** Создайте новый прямоугольник и сделайте его перетаскиваемым. При нажатии на него и последующим движении мыши он должен начать двигаться вместе с курсором. При отпускании мыши должен остаться на месте.

- **Fullscreen:** Чтобы перейти в fullscreen нужно установить стиль `sf::RenderWindow` как `Fullscreen` вместо `Default`.

- **Текст:** В папке `3text` содержится пример для работы с текстом. Сделайте так, чтобы при нажатии клавиши пробел у текста задавалась случайная позиция, случайный поворот, случайный цвет и случайное масштабирование(в разумных пределах).

- **Печать чисел:** Создайте новый текст, который будет печатать координаты мыши и меняться при движении мыши. Для перевода чисел в строку используйте функцию `std::to_string`.

- **Столкновение шаров:** В папке `4collision_circles` содержится заготовка кода. Используйте этот код, чтобы найти пересечение двух шаров. Если в процессе движения шары начнут накладываются друг на друга, то они должны окрашиваться в красный цвет. После прекращения наложения, шары должны опять стать белыми. Для этого добавьте поле типа `sf::Color` в класс `Ball` и метод `bool is_colliding(const Ball& b) const`, который будет проверять 2 кружка на столкновение.
- **Отражение шаров:** Измените программу так, чтобы кружки упруго отскакивали друг от друга. Для этого нужно, при столкновении шариков, обратить составляющую скорости параллельную прямой, соединяющую центры шариков.
- **Много шаров:** Создайте 10 шариков, которые будут сталкиваться друг с другом.
- **Добавление шаров:** Добавьте возможность добавления шаров по нажатию кнопки мыши.
- **Pong:** Создайте игру Pong на 2 игрока. Первый игрок должен управлять ракеткой используя клавиши `W` и `S`. Второй игрок – стрелочки вниз и вверх.