

# Руководство пользователя

## Общее описание

Алгоритм работы фреймворка хорошо показана на данной схеме.



Сначала вызывается функция simulate класса Dynamic, которая последовательно считывает, моделирует и выводит, используя наследников классов VRead, VUpdate и VWrite соответственно. Фреймворк состоит из 3 абстрактных классов, реализующий интерфейсы классов чтения, обновления и записи: VRead, VUpdate, VWrite, 3 класса контейнера: SystemState, Particle, Vector3, 2 заголовочных файлов: headers.h с описанием скорости света и подключением нужных библиотек и functions.h с описанием функций фабрик для генерации сил, а так же 1 классом менеджером - Dynamic, управляющий процессами в работе фреймворка.

## Интерфейсные классы

Всего в фреймворке описано 3 интерфейсных класса.

### VRead - модуль для чтения

В классе *VRead* определены 3 функции:

- ввод вектора - без параметров, возвращает *Vector3*
- ввод частицы - без параметров, возвращает *Particle*
- ввод состояния системы - принимает кол-во ввозимых частиц, возвращает *SystemState*

## **VUpdate - модуль для обновления**

В классе *VUpdate* определена 1 функция для обновления частиц - *updateSystemState*, принимает на вход состояние системы, внешнюю силу, силу взаимодействия 2 частиц и промежуток времени. Очень важно, чтобы промежуток времени был достаточно мал. Функция возвращает новое состояние системы.

## **VWrite - модуль для вывода**

В классе *VWrite* определены 3 функции:

- вывод вектора - принимает *Vector3*, ничего не возвращает
- вывод частицы - принимает *Particle*, ничего не возвращает
- вывод состояния системы - принимает *SystemState*, ничего не возвращает

# **1. Стандартные реализации**

Все реализации лежат в директории `implem`

В директории они лежат в 3 папках

## **Read**

Для *Vread* есть 3 стандартные реализации

## **StdCinRead**

---

Использует стандартный поток ввода `stdin`. Только конструктор по умолчанию. Все числа вводятся с разделителями: пробел или перенос строки.

- Вектор считывает, как 3 числа с плавающей.
- Частицу читает, как 2 вектора: скорость и координаты и 2 числа с плавающей: масса и спин.
- Вводит состояние системы, как текущее время: одно число с плавающей точкой, а после этого *num\_of\_particles* частиц.

## StreamRead

---

Использует стандартные потоки `std::istream`. Только конструктор от `std::istream`. В конструкторе происходит копирование потока. Все числа вводятся с разделителями: пробел или перенос строки.

- Вектор считывает, как 3 числа с плавающей точкой.
- Частицу читает, как 2 вектора: скорость и координаты и 2 числа с плавающей точкой: масса и спин.
- Вводит состояние системы, как текущее время: одно число с плавающей точкой, а после этого *num\_of\_particles* частиц.

## FileRead

---

Использует файловые потоки `std::ifstream`. Два конструктора

1. Конструктор от пути до файла: происходит открытие файла экземпляром класса *FileRead*.

2. Конструктор от *std::ifstream*: происходит копирование потока. Все числа вводятся с разделителями: пробел или перенос строки.
- Вектор считывает, как 3 числа с плавающей точкой.
  - Частицу читает, как 2 вектора: скорость и координаты и 2 числа с плавающей точкой: масса и спин.
  - Вводит состояние системы, как текущее время: одно число с плавающей точкой, а после этого *num\_of\_particles* частиц.

## Update

Для класса *VUpdate* есть одна реализация - *UpdateNoCollisions*. Предоставляет стандартное обновление частиц, без учета их столкновений.

Обновление происходит по следующей схеме:

## Расчет сил

---

В самом начале считаются силы, действующие на каждую частицу. Сила, действующая на частицу - это сумма сил, действующие между частицами, плюс внешняя сила. Силы считаются квадратным алгоритмом: во внешнем цикле перебираются частицы, для которых мы считаем силу, а во внутреннем, через которых считаем. Силы между двумя частицами записываются в вектор сил *std::vector<Vector3> particles\_forces\_array*. Если индекс в первом цикле равен индексу во втором, то в *particles\_forces\_array* остается значение по умолчанию (конструктор по умолчанию вектора 3).

Далее сумма всех сил записываются в *std::vector<Vector3> forces* посредством линейного прохода для дальнейшего использования.

# Обновление системы

---

Сначала создается новое, изначально равное старому. Это сделано для того, чтобы все вектора сразу приняли нужный размер. `

Далее линейным проходом пересчитываются координаты и скорость

Координаты считаются через старую, еще не обновленную скорость. Пересчет идет по формуле

$x = coordinates.x + velocity.x * dt$ , где

- $x$  - одна из 3 координат, пересчет ведется по всем
- *coordinates* - координаты частицы
- *velocity* - скорость частицы
- $dt$  - промежуток времени

Скорость пересчитывается через релятивистский закон Ньютона в динамике.

После обновления координат в новой системе обновляется время и она возвращается как результат функции.

## Write

Для VWrite есть 3 стандартные реализации

### StdCoutWrite

---

Использует стандартный поток вывода stdout. Только конструктор по умолчанию.

- Выводит вектор, как 3 числа числа с плавающей точкой, разделенными пробелами
- Выводит частицу, как 2 вектора: скорость и координаты и два числа с плавающей точкой: масса и спин.
- Вводит состояние системы, как текущее время: одно число с плавающей точкой, а после этого *num\_of\_particles* частиц.

## StreamWrite

---

Использует стандартные потоки *std::ostream*. Только конструктор от *std::ostream*. В конструкторе происходит копирование потока.

- Выводит вектор, как 3 числа числа с плавающей точкой, разделенными пробелами
- Выводит частицу, как 2 вектора: скорость и координаты и два числа с плавающей точкой: масса и спин.
- Вводит состояние системы, как текущее время: одно число с плавающей точкой, а после этого *num\_of\_particles* частиц.

## FileWrite

---

Использует файловые потоки *std::ofstream*. Два конструктора

1. Конструктор от пути до файла: происходит открытие файла экземпляром класса *FileWrite*.
  2. Конструктор от *std::ofstream*: происходит копирование потока.
- Выводит вектор, как 3 числа числа с плавающей точкой, разделенными пробелами

- Выводит частицу, как 2 вектора: скорость и координаты и два числа с плавающей точкой: масса и спин.
- Вводит состояние системы, как текущее время: одно число с плавающей точкой, а после этого *num\_of\_particles* частиц.

## 2. Свои реализации

Для любого из интересных классов есть возможность написать свои реализации. Для этого достаточно унаследовать от них и переопределить все функции.

## Классы контейнеры

### Vector3 - трехмерный вектор

*Vector3* представляет собой трехмерный вектор.

В классе определены следующие конструкторы:

- Конструктор по умолчанию: все параметры будут заполнены нулями
- Конструктор от 3 переменных типа *double*: копирование этих переменных в параметры вектора
- Копирующий конструктор: поэлементное копирование параметров

### Particle - частица

Класс *Particle* описывает состояние одной частицы

В классе описаны 4 поля:

- Скорость частицы (*velocity\_*). Тип *Vector3*
- Координаты частицы (*coordinates\_*). Тип *Vector3*

- Масса частицы (*mass\_*). Тип *double*
  - Изоспин частицы (*I\_*), задает тип частицы. Тип *float*
- Определены следующие конструкторы:
- Конструктор по умолчанию: *mass\_* и *I\_* нули, *velocity\_* и *coordinates\_* - конструкторы по умолчанию
  - Конструктор от 4 переменных: *Vector3 velocity*, *Vector3 coordinates*, *double mass*, *float I*. Копирование *velocity* -> *velocity\_*, *coordinates* -> *coordinates\_*, *mass* -> *mass\_*, *I* -> *I\_*
  - Конструктор копирования: поэлементное копирование

Так же в классе *Particle* описана булевая функция *isNormalVelocity()*, она не принимает параметров и возвращает *true*, если все измерения скорости строго меньше скорости света.

## SystemState - состояние системы

Класс *SystemState* описывает текущее состояние системы

Класс хранит 2 параметра:

- Текущее время (*time\_*). Тип *double*
- Вектор частиц (*particles\_*). Тип *std::vector<Particle>*

Определены следующие конструкторы:

- Конструктор по умолчанию: *time\_* равно нулю, *particles\_* размера 0;
- Конструктор от 2 переменных: *double time*, *std::vector<Particle> particles*. Копирование *time* -> *time\_*, *particles* -> *particles\_*
- Конструктор копирования: поэлементное копирование

Для класса перегружены константный и неконстантный операторы квадратные скобки для индексирования по вектору *particles\_*. При



выходе за границы массива программа упадет с сообщением "Array of particles in SystemState.h out of range"

Для класса определена функция *size()*, которая возвращает размер вектора *particles\_*

## Dynamic - класс менеджер

Dynamic - это класс менеджер, контролирующий работу всего фреймворка.

Он хранит в себе:

- Состояние системы (*state\_*). Тип *SystemState*
- Внешнюю силу, действующую на частицы (*external\_f\_*). Тип *std::function<Vector3(Particle)>*
- Силу, действующую между двумя частицами (*f\_btw\_two\_par\_*). Тип *std::function<Vector3(Particle, Particle)>*
- Указатель на модуль чтения (*reader\_*). Тип *std::unique\_ptr<VRead>*
- Указатель на модуль обновления (*updater\_*). Тип *std::unique\_ptr<VUpdate>*
- Указатель на модуль записи (*writer\_*). Тип *std::unique\_ptr<VWrite>*

У него есть только один конструктор: на вход подаются:

- *std::unique\_ptr<VRead> reader*
- *std::unique\_ptr<VUpdate> updater*
- *std::unique\_ptr<VWrite> writer*
- *std::function<Vector3(Particle)> external\_f*
- *std::function<Vector3(Particle, Particle)> f\_btw\_two\_par*

В конструкторе проходит копирование сил и передача

указателей

## **simulate - метод симуляции**

В *Dynamic* определен метод *simulate*, предназначенный для симуляции движения частиц. Метод получает время симуляции, разбивает его на заданное количество промежутков (промежутки по времени должны быть маленькими) и проводит обновление частиц за все эти промежутки.

На вход метод принимает 4 параметра:

- *num\_of\_iterations* - количество итераций обновления. Тип *int*
- *time* - общее время движения частиц. Тип *double*
- *num\_of\_particles* - количество частиц. Тип *size\_t*
- *save\_states* - флаг, нужно ли сохранять промежуточные значения системы. Тип *bool*

В начале считывается состояние системы.

После этого проходит проверка, что в *state\_* все частицы корректные, с помощью метода *isNormalVelocity()* класса *Particle*. Если метод *isNormalVelocity()* вернул *false*, то программа завершает свою работу с ошибкой "The speed of a particle is greater than the speed of light".

Если функция продолжила свою работу, то дальше идут итерации обновления частиц. Проходит *num\_of\_iterations* итераций, на каждой из которых выполняется обновление и идет проверка на корректность скоростей частиц. Затем, если *save\_states* равен *true*, идет запись в вектор состояний системы.

В конце, когда все итерации пройдены, в зависимости от переменной *save\_states* выводятся либо все состояния системы,

либо только конечное.

# Интерфейс для представления сил

Для представления сил было решено использовать *std::function*, возвращающий *Vector3* - силу. Есть два вида сил: взаимодействие двух частиц и постоянная сила. Первая описывается с помощью *std::function<Vector3(Particle, Particle)>* (на вход идут 2 частицы, на выходе *Vector3*). Вторая описывается с помощью *std::function<Vector3(Particle)>* (на вход идет одна частица, на выходе *Vector3*).

## 1. Стандартные реализации для сил

Стандартные реализации представляют собой функции-фабрики. На вход им подаются различные константы, а на выходе вы получаете *std::function<Vector3(Particle, Particle)>* либо *std::function<Vector3(Particle)>*.

В данный момент реализовано 2 функции для силы между частицами (возвращают *std::function<Vector3(Particle, Particle)>*):

- сила Кулона
  - сила Юкавы
- И две внешние силы (возвращают *std::function<Vector3(Particle)>*):
- постоянное электрическое поле
  - постоянное магнитное поле

## 2. Добавление собственной силы

Для добавления своей силы достаточно создать *std::function* с нужными параметрами и передать их классу *Dynamic*

- `std::function<Vector3(Particle, Particle)>` для взаимодействия двух частиц
- `std::function<Vector3(Particle)>` для внешней силы