```
1) CPP11
// В стратегию приходят ссылки четыре массива
std::vector<Elevator>& myElevators;
std::vector<Passenger>& myPassengers;
std::vector<Elevator>& enemyElevators;
std::vector<Passenger>& enemyPassengers;

// Для отладки в консоль используйте вызов this->log(T smth)
// Допустимы типы: bool, int, short, long, float, double, char, QString, Passenger,
Elevator

// Каждый тик вызывается strategy->onTick(List<Passenger>, List<Elevator>,
List<Passenger>, List<Elevator>)

// API для работы с пассажирами
class Passenger {
public:
    // Текущий этаж
    int floor;
    // Этаж, с которого едет пассажир
    int from_floor;
    // Этаж, на который едет пассажир
    int dest_floor;
    // Координаты пассажира
    int x, y;
    // Сколько времени осталось до ухода на лестницу
    int time_to_away;
    // Узнать к какому игроку привязан пассажир
    // Возвращаемое значение "FIRST_PLAYER", "SECOND_PLAYER"
    QString type;

public:
    // проверить, есть ли у пассажира лифт
    bool has_elevator() {}
    // назначить пассажиру лифт
    void set_elevator(const Elevator& elevator) {}
};

// API для работы с лифтами
class Elevator {
public:
    // Текущий этаж
    int floor;
    // Этаж, к которому лифт едет в данный момент
    int next_floor;
    // Сколько времени лифт простоял на этаже
    int time_on_floor;
    // Координата лифта по Y
    double y;
    // Скорость движения в текущий момент
    double speed;
    // Какому игроку принадлежит
    // Возвращаемое значение "FIRST_PLAYER", "SECOND_PLAYER"
    QString type;

    // Массив пассажиров, которых перевозит данный лифт
    std::vector<Passenger> passengers;

public:
    // Отправить лифт на указанный этаж, он доедет и выпустит пассажиров
    void go_to_floor(int floor) {}
};
```

2) Java 1.8 + Oracle JDK

```
// В стратегию приходит четыре массива
ArrayList<Elevator> myElevators;
ArrayList<Passenger> myPassengers;
ArrayList<Elevator> enemyElevators;
ArrayList<Passenger> enemyPassengers;

// Для отладки в консоль используйте вызов
this.log(Object object)

// Каждый тик вызывается Strategy.onTick(List<Passenger>, List<Elevator>,
List<Passenger>, List<Elevator>)

// API для работы с пассажирами
class Passenger {
    // Назначить пассажиру лифт
    public void setElevator(Elevator elevator) {}
    // Проверяет, назначен ли лифт пассажиру
    public Boolean hasElevator() {}
    // Текущий этаж
    public Integer getFloor() {}
    // Этаж, с которого едет пассажир
    public Integer getFromFloor() {}
    // Этаж, на который едет пассажир
    public Integer getDestFloor() {}
    // Сколько времени осталось до ухода на лестницу
    public Integer getTimeToAway() {}
    // Координаты пассажира
    public Double getY() {}
    public Double getX() {}
    // Узнать к какому игроку привязан пассажир
    // Возвращаемое значение "FIRST_PLAYER", "SECOND_PLAYER"
    public String getType() {}
}

// API для работы с лифтами
class Elevator {
    // Отправить лифт на указанный этаж, он доедет и выпустит пассажиров
    public void goToFloor(Integer floor) {}
    // Массив пассажиров, которых перевозит данный лифт
    public List<Passenger> getPassengers() {}
    // Текущий этаж
    public Integer getFloor() {}
    // Этаж, к которому лифт едет в данный момент
    public Integer getNextFloor() {}
    // Сколько времени лифт простоял на этаже
    public Integer getTimeOnFloor() {}
    // Координата лифта по Y
    public Double getY() {}
    // Скорость движения в текущий момент
    public Double getSpeed() {}
    // Какому игроку принадлежит
    // Возвращаемое значение "FIRST_PLAYER", "SECOND_PLAYER"
    public String getType() {}
}
```

```
3) JavaScript + NodeJS 7.4

// В стратегию приходит четыре массива
Array из Elevator myElevators
Array из Passenger myPassengers
Array из Elevator enemyElevators
Array из Passenger enemyPassengers

// Для отладки в консоль используйте вызов
this.debug(String)
// Каждый тик вызывается Strategy.onTick(myPassengers, myElevators,
enemyPassengers, enemyElevators)
// API для работы с пассажирами
class Passenger {
    // Назначить пассажиру лифт
    setElevator(elevator) {}
    // Проверяет, назначен ли лифт пассажиру :Boolean
    hasElevator() {}

    // Текущий этаж :Number
    get floor() {}
    // Этаж, с которого едет пассажир :Number
    get fromFloor() {}
    // Этаж, на который едет пассажир :Number
    get destFloor() {}
    // Сколько времени осталось до ухода на лестницу :Number
    get timeToAway() {}
    // Координаты пассажира :Number
    get x() {}
    get y() {}
    // Узнать к какому игроку привязан пассажир :String
    // Возвращаемое значение "FIRST_PLAYER", "SECOND_PLAYER"
    get type() {}
}

// API для работы с лифтами
class Elevator {
    // Отправить лифт на указанный этаж, он доедет и выпустит пассажиров :Number
    go_to_floor(floor) {}
    // Массив пассажиров, которых перевозит данный лифт :Array
    get passengers() {}

    // Текущий этаж :Number
    get floor() {}
    // Этаж, к которому лифт едет в данный момент :Number
    get nextFloor() {}
    // Сколько времени лифт простоял на этаже :Number
    get timeOnFloor() {}

    // Координата лифта по Y :Number
    get y() {}
    // Скорость движения в текущий момент :Number
    get speed() {}
    // Какому игроку принадлежит :String
    // Возвращаемое значение "FIRST_PLAYER", "SECOND_PLAYER"
    get type() {}
}
```

4) PHP7

```php
// В стратегию приходит четыре массива
$my_passengers: array из Passenger
$my_elevators: array из Elevator
$enemy_passengers: array из Passenger
$enemy_elevators: array из Elevator
// Для отладки в консоль используйте вызов
$this->log($text)
// Каждый тик вызывается Strategy->on_tick($my_passengers, $my_elevators,
$enemy_passengers, $enemy_elevators)
// API для работы с пассажирами
class Passenger {
    function __construct() {
        // Текущий этаж :integer
        $this->floor = $floor;
        // Этаж, с которого едет пассажир :integer
        $this->from_floor = $from_floor;
        // Этаж, на который едет пассажир :integer
        $this->dest_floor = $dest_floor;
        // Сколько времени осталось до ухода на лестницу :integer
        $this->time_to_away = $time_to_away;
        // Координаты пассажира :float
        $this->x = $x;
        $this->y = $y;
        // Узнать к какому игроку привязан пассажир :string {"FIRST_PLAYER",
"SECOND_PLAYER"}
        $this->type = $type;
    }
    // Назначить пассажиру лифт
    public function set_elevator($elevator) {}
    // Проверяет, назначен ли лифт пассажиру
    public function has_elevator() : bool {}
}

// API для работы с лифтами
class Elevator {
    function __construct() {
        // Текущий этаж :integer
        $this->floor = $floor;
        // Этаж, к которому лифт едет в данный момент :integer
        $this->next_floor = $next_floor;
        // Сколько времени лифт простоял на этаже :integer
        $this->time_on_floor = $time_on_floor;
        // Координата лифта по Y :float
        $this->y = $y;
        // Скорость движения в текущий момент :float
        $this->speed = $speed;
        // Какому игроку принадлежит :string {"FIRST_PLAYER", "SECOND_PLAYER"}
        $this->type = $type;
        // Массив пассажиров, которых перевозит данный лифт
        $this->$passengers = $passengers;
    }
    // Отправить лифт на указанный этаж, он доедет и выпустит пассажиров :integer
    public function go_to_floor(floor) {}
}
```

```
5) Python 2.7 (Python 3.6 аналогично)
# В стратегию приходит четыре массива
list из Elevator: my_elevators
list из Passenger: my_passengers
list из Elevator: enemy_elevators
list из Passenger: enemy_passengers
# Для отладки в консоль используйте вызов
self.debug(str)
# Каждый тик вызывается Strategy.on_tick(my_elevators, my_passengers,
enemy_elevators, enemy_passengers)
# API для работы с пассажирами
class Passenger(object):
    def __init__(self, x, y, type, time_to_away, from_floor, dest_floor, floor):
        # Текущий этаж :int
        self.floor = floor
        # Этаж, с которого едет пассажир: int
        self.from_floor = from_floor
        # Этаж, на который едет пассажир :int
        self.dest_floor = dest_floor
        # Сколько времени осталось до ухода на лестницу :int
        self.time_to_away = time_to_away

        # Координаты пассажира :float
        self.x, self.y = x, y
        # Узнать к какому игроку привязан пассажир :string
        # Возвращаемое значение "FIRST_PLAYER", "SECOND_PLAYER"
        self.type = type
    # Проверяет, назначен ли лифт пассажиру
    def has_elevator(self): pass
    # Назначить пассажиру лифт
    def set_elevator(self, elevator): pass

# API для работы с лифтами
class Elevator(object):
    def __init__(self, y, passengers, speed, floor, next_floor, time_on_floor,
type):
        # Текущий этаж :int
        self.floor = floor
        # Этаж, к которому лифт едет в данный момент :int
        self.next_floor = next_floor
        # Сколько времени лифт простоял на этаже :int
        self.time_on_floor = time_on_floor
        # Координата лифта по Y :float
        self.y = y
        # Скорость движения в текущий момент :float
        self.speed = speed
        # Какому игроку принадлежит :string
        # Возвращаемое значение "FIRST_PLAYER", "SECOND_PLAYER"
        self.type = type
        # Массив пассажиров, которых перевозит данный лифт :list из Passenger
        self.passengers = passengers
    # Отправить лифт на указанный этаж :int, он доедет и выпустит пассажиров
    def go_to_floor(self, floor): pass
```