Разработка компьютерной модели ЗРК

Управляющий модуль

Класс Simulation

Описание: главный класс моделирования. Реализует создание всех базовых классов и их обновление с шагом моделирования.

Поля:

- Dispatcher
- Gui
- skyEnv
- CC
- db
- steps – количество шагов моделирования

Методы:

- set_dispatcher()
- set_GUI() –
- set_units() – создаёт экзепляры класса SkyEnv и CC.
- modulate – функция, вызывающая метод update у базовых классов в цикле по количеству шагов

Класс DatabaseManager

Описание: класс для записи и хранени информации от пользователя по параметрам моделирования.

Поля:

- name
- cursor
- conn

Методы:

- set_name() - добавление имени бд

- create_tables() – создание таблиц
- add_plane() – добавление самолёта
- add_launcher()-добавление ПУ
- add_radar()- добавление радара
- load_planes()
- load_launchers()
- load_radars()
- close()

Технологии

SQLite используется для создании и записи в бд.

Модуль диспетчер

Назначение: связь между модулями для передачи информации при помощи объектов класса Message.

Класс Dispatcher

Описание: получение объектов класса Message, их хранение и пересылка между ВО, ПБУ, РЛС, ПУ и GUI.

Поля

- **currentTime** – шаг моделирования
- **messageQueues** – словарь, хранящий пары: название модуля - список с сообщениями, которые отправили модулю за шаг времени.
- **databasemanager** – объект класса DatabaseManager

Методы

- **register(recipient_id)** - добавление ключа в словарь, создание пустой очереди для модуля
- **send_message(message)** - отсылка сообщения. Тип данных объект класса Message. При вызове диспетчер добавляет сообщение в очередь получателя
- **get_message(recipient_id)** - возвращает сообщения получателю за предыдущий шаг моделирования в порядке приоритета. Сообщения - объекты класса Message.
- **params_for_db()** – записывает данные в бд.

- **planes_for_SE()** – считывает из базы данных данные о самолётах и отправляет их SkyEnv.
- **radars_and_launchers_for_CC()** - считывает из базы данных данные о ПУ, РЛС и ЗУР и отправляет их ControlCenter.

Класс Message(dataclass)

Описание: класс для записи и хранени информации от отправителя получателю

Поля:

- **recipient_id()** – id модуля – получателя (GUI, RadarMain, SE, LauncherMain, ControlCenter)
- **priority –** приоритет сообщения в очереди (LOW, STANDARD, HIGH)

Классы – наследники:

1. SEStarting

Поля:

- **GUI, RadarMain**
- **LOW**
- **planes** – словарь пар: id_plane – его траектория в формате np.array([x,y,z].

2. SEKilled

Поля:

- **GUI, RadarMain**
- **HIGH**
- **collisionStep** – шаг моделирования.
- **rocketId**
- **rocketCoords**
- **planeId**
- **planeCoords**
- **collateralDamage** – список объектов, которые оказлись в зоне поражения и были случайно уничтожены.

3. SEAddRocket

Поля:

- **GUI**
- **STANDARD**

- startTime
- rocketId
- rocketCoords

4. SEAddRocketToRadar

Поля:

- RadarMain
- STANDARD
- startTime
- missile
- rocketCoords

5. CCLaunchMissile

Поля:

- LauncherMain
- HIGH
- target

6. CCToRadarNewStatus

Поля:

- RadraMain
- STANDARD
- target_new_status

7. CCToSkyEnv

Поля:

- SkyEnv
- STANDARD
- missiles

8. RadarToGUICurrentTarget

Поля:

- GUI
- HIGH
- radarId
- targetId
- sectorSize

9. RadarControllerObjects

Поля:

- **ControlCenter**
- **STANDARD**
- **detectedObjects**

10. LaunchertoSEMissileLaunched

Поля:

- **SkyEnv**
- **STANDARD**
- **targetId**
- **missile**

11. LauncherToCCMissileLaunched

Поля:

- **ControlCenter**
- **STANDARD**
- **Missile**

12. ToGuiRocketInactivated

Поля:

- **GUI**
- **HIGH**
- **rocketId**

Модуль отображения результатов моделирования

Назначение:

Модуль обеспечивает интерфейс для ввода данных моделирования, выполняет отображения результатов моделирования:

- отображение информации о воздушной обстановке;
- отображение текущей позиции ЗРК на карте;
- отображение статуса ПУ, ЗУР;
- отображение процесса моделирование (текстовое описание событий моделирования) .

Класс StartPage

Поля:

- stepsNumber - число шагов моделирования
- currentStep - текущий шаг моделирования

Методы:

- **set_session_params(db) – создание стартовой страницы для ввода данных от пользователя. Возвращает количество шагов моделирования.**
- **display_current_state(process_desc,zur_state,rls_state, delete_object)** - отображение текущего состояния системы, шага моделирования
- **visualize_plane_track(planeId, coord) -** визуализация траекторий самолетов
- **visualize_rls_sector() -** визуализация области обзора радиолокатора
- **visualize_zur_track(zur_id, coord, detection_area)** - визуализация полета ЗУР, области поражения цели зенитно-управляемой ракетой

Технологии

QtCreator, PyQt: **реализация графического интерфейса пользователя (GUI)**

Folium: **отображение интерактивных графических карт**

Модуль Воздушной обстановки

Назначение: создание и контроль объектов в небе.

Класс SkyEnv

Описание: моделирует воздушную среду с самолетами и ракетами, обрабатывая их траектории, столкновения и взаимодействия. Интегрируется с системой диспетчеризации сообщений для коммуникации с другими модулями.

Поля:

- **planes** — список объектов класса Plane

- **rockets** — словарь объектов класса Rocket {rocket_id, Rocket}
- **paires** — словарь из пар: rocketID - plane
- **currentTime** — шаг моделирования

- **dispatcher** — объект класса Dispatcher
- **killedLastStep** - список уничтоженных объектов
- **timeSteps** - общее время симуляции
- **to_remove** - множество объектов на удаление
- **id** - идентификатор модуля (6)

Методы:

- **make_planes** — создаёт объекты класса Plane и добавляет их в planes.
- **check_collision** — проверяет столкновение целей
- **check_if_in_radius** — проверка попадания других объектов в область взрыва
- **remove_plane** - убрать самолёт из списка planes
- **add_rocket** - добавить раекту в список rockets
- **remove_rocket** - удалить ракету из rockets
- **add_pair** - добавить пару rocketID - plane в paires
- **delete_pair** - удалить пару rocketID - plane из paires
- **start** – регистрация у дичпетчера, рассчёт всех траекторий самолётов и рассылка сообщений SEStarting для GUI и RadarController
- **update** - обновить все объекты в списке rockets и planes и отослать необходимые сообщения другим модулям

Вспомогательные функции:

- **get_plane_trajectory_from_rocket(paires, rocket)** - Возвращает траекторию самолета-цели для ракеты.
- **get_plane_id_from_rocket(paires, rocket)** -Возвращает ID самолета-цели для ракеты.

- **vector(a,b)** - Вычисляет вектор между точками.
- **distance(a,b)** - Вычисляет расстояние между точками.

Класс SkyObject

Описание: класс-родитель для классов Plane и Rocket. Абстрактное представление объекта в воздушном пространстве с базовым расчетом траектории.

Поля:

- **id** — идентификатор
- **currentPos**— координаты в данный момент времени.
- **start** — начальные координаты.
- **finish** — конечные координаты.
- **trajectory** — список всех координат за период моделирования по шагам.
- **timeSteps** - общее количество шагов симуляции.
- **currentTime** - текущий шаг симуляции.

- **speed** - скорость движения самолёта = 500.

Методы:

- **calculate_trajectory** — рассчитывает **trajectory**
- **update** — обновляет позицию.
- **get_id** — возвращает id.
- **get_trajectory** - возвращает **trajectory.**
- **get_currentPos** — возвращает позицию в этот шаг.
- **get_speed** - возвращает скорость.

Класс Plane

Описание: Представляет самолет с реалистичной траекторией полета.

Поля:

- **status** — состояние самолёта.
- **points** – дополнительные точки траектории от пользователя (в текущей реализации не используется).

Методы:

- **get_status** - получение состояния самолёта
- **calculate_trajectory()** - переопределяет родительский метод, добавляя фазы полета (набор высоты, крейсерский полет, снижение).

- **killed** - помечает самолет как уничтоженный.

Класс Rocket

Описание: созданная ракета, отслеживающая самолёт

Поля:

- **lifePeriod** — период жизни ракеты
- **killed** – состояние ракеты
- **radius** — радиус поражения ракеты

- **velocity** - вектор скорости (массив NumPy)

- **startTime** – время взлёта
- **dragcoeff** - коэффициент сопротивления (число с плавающей точкой)
- **gravity** - ускорение свободного падения (число с плавающей точкой)

Методы:

- **get_radius** – возвращает радиус поражения
- **calculate_trajectory()** - вычисляет траекторию с учетом физики полета
- **get_startTime** – возвращает время начала полёта
- **boom** – помечает ракету взорвавшейся
- **is_killed** – возвращает данные о состоянии

Модуль ПБУ

Назначение: моделирование работы ПБУ, т.е координация действий РЛС, ПУ и ЗУР.

Класс ControlCenter

Описание: Класс ControlCenter является центральным элементом системы, который управляет всеми компонентами: радарами, пусковыми установками и ракетами. В нем хранятся контроллеры для каждого типа устройства и ссылки на диспетчер сообщений.

Поля:

- **radarController** — объект класса RadarController, отвечающий за управление радарами.

- **launcherController** — объект класса LauncherController, отвечающий за управление установками для запуска ракет.
- **missileController** — объект класса MissileController, отвечающий за управление ракетами.
- **dispatcher** — объект класса Dispatcher, который используется для отправки и получения сообщений**.**
- **position** – расположение ПБУ
- **targets** — словарь, хранящий все приследуемые цели (самолеты).

Методы:

- **update()** — метод, вызываемый на каждой итерации цикла для получения и обработки сообщений. Также вызывает обновление состояния радаров, пусковых установок и ракет.

- **get_position()** – возвращает расположение ПБУ
- **get_targets()** — возвращает список всех известных целей.
- **get_launchers()** — возвращает список всех пусковых установок (установок для запуска ракет).
- **get_radars()** — возвращает список всех радаров.
- **get_radar_controller()** — возвращает объект RadarController.
- **get_launcher_controller()** — возвращает объект LauncherController.

- **get_missiles(self)** - возвращает список всех ракет (одноразово)
- process_targets – обрабатывает цели
- **update_proirity_targets** – меняет приоритет целям на новой итерации
- **current_priority_targets** – находит старые приоритеты цели на прошлой итерации
- **find_priority_targets** – находит приоритетные цели на данной итерации
- **direction** – вычислет единичный вектор направления на target

Модуль РЛС

Назначение: моделирование работы радиолокационной станции.

Класс Radar

Описание: Класс Radar представляет собой объект радара, который фиксирует объекты в своей зоне видимости и передает данные в RadarController.

Поля:

- **position (tuple)** - координаты радара в глобальной системе (X, Y, Z)
- **range (float)** - радиус/область зоны сканирования.
- **detectedObjects (list)** - список обнаруженных объектов с их относительными координатами.
- **radarId (str)** - номер или уникальный идентификатор радара.
  Пример: "Radar-001" — идентификатор радара.
- **maxTargetCount (int)** - число одновременно сопровождаемых целей, которое возможно.
- **currentTargetCount (int)** - количество целей, которые радар в данный момент сопровождает.
- **noiseLevel (float)** - параметр внутренних шумов радара.
- **detectedMissiles (list)** - список обнаруженных ракет с их относительными координатами.

Методы:

- **scan()** - выполняет сканирование области и обновляет список detectedObjects.
- **get_detected_objects() -> list**: Возвращает список обнаруженных объектов.
- **get_detected_missiles() -> list**: Возвращает список ракет.
- **track_target(targetId: str)**: метод, который реализует сопровождение цели с более узким радиусом слежения и более высокой точностью.
- **mark_target_as_destroyed(targetId: str)**: Метод, который помечает цель как уничтоженную

Класс RadarContoller

Описание: класс RadarController управляет несколькими радарами, агрегирует данные и передает их в ControlCenter.

Поля:

- **radars** - словарь всех подключенных радаров.

- **dispatcher**
- **all_targets**
- **detected_targets**
- **all_missiles**

Методы:

- **add_radar** - добавляет радар в систему.
- **process_radar_data()** - запрашивает данные со конкретного радара, преобразует координаты в абсолютные и отправляет в ControlCenter.
- **instruct_radar_to_track_target(radarId: str, targetId: str)**: Метод, который дает конкретному радару указание отслеживать цель с заданным ID. В этом методе отсылается сообщение RadarToGUICurrentTarget.
- **process_message** – обработка полученных сообщений
- **update** – вызывает функции обновления от всех радаров. Получение и отправка сообщений

Класс Target

Описание: Класс Target - это цель, обнаруженная радаром и отслеживаемая им. По команде ПБУ, ПУ выпускает по ней ракету

Поля:

- **targetID** — уникальный идентификатор цели также является идентификатором самолёта, который и является целью
- **radarID** — идентификатор радара, сопровождающего цель
- **isFollowed** — статус цели: были ли по нему выпущены ракеты.
- **currentPosition** — положение цели по данным радара.
- **missilesFollow** — список ракет, направленных на уничтожение
- **status** – статус объекта
- **clear_coords** - реальные координаты, полученные от SE
- **currentSpeedVector** – текущий вектор скорости
- **currentCoords** – текцщие координаты цели

Методы:

- **attach_missile**— добавляет ракету, направленную на цель
- **detach_missile** – удаляет ракету из списка привязанных
- **update_current_coords** – обновляет текущие координаты
- **update_speed_vector** – обновляет текущий вектор скорости
- **update_status** – обновляет статус объекта
- **get_missilesFollow** — возвращает список всех ракет
- **get_targetID** — возвращает идентификатор цели

Модуль ПУ

Назначение: моделирование работы ПУ

Класс LaunchController

Описание: Объект хранит данные о пусковых установках (их местоположении и боеприпасах), а также отправляет сообщения о запуске.

Поля:

- **launchers** — список объектов класса Launcher

Методы:

- **update()** — производится каждый цикл, проверяет наличие новых сообщений и обрабатывает их.
- **create()** — в случае получения информации о самолёте выбирает пусковую установку и вызывает у неё launch(), передаёт данные ракеты на ПБУ (получает сообщение CCLaunchMissile)
- **status()** — сообщает о состоянии боекомплекта в каждой из ПУ;

Класс Launcher

Описание: Отдельная пусковая установка.

Поля:

- **silo_num** — информация о заполненности пусковых шахт.
- **coord** — координаты пусковой установки.

Методы:

- **launch(target)** — создаёт объект Missile и добавляет его в target с помощью функции **attach_missile** . Отсылает сообщения LaunchertoSEMissileLaunched и LauncherToCCMissileLaunched
- **status()** — сообщает состояние боекомплекта.

Модуль ЗРУ

Класс MissileController

Описание: Класс MissileController управляет всеми ракетами в системе. Он обновляет состояние ракет.

Поля:

- **missiles** —все ракеты, направленные на перехват целей на этой итерации
- **unusefulMissiles** – ненужные ракеты на данной итерации

Методы:

- **process_missiles_of_target** - обрабатывает список ракет у данной цели
- **process_unuseful_missiles** - обрабатывает список всех ненужных ракет
- **process_new_missile** **-** обрабатывает новую ракету
- **pop_missiles** - удаляет и возвращает список всех ракет
- **destroy_missile** **-** уменьшает счетчик времени жизни ракеты до нуля
- **collision** - проверяет наличие объекта в радиусе объекта
- **will_explode** - проверяет, что target будет в радиусе взрыва ракеты missile

Класс Missile

Описание: Созданная ракета

Поля:

- **status** — состояние ракеты
- **missileId** — уникальный идентификатор ракеты

- **targetId -** уникальный идентификатор цели
- **currentPosition** — положение ракеты по данным радара, в момент запуска равно положению Launcher, который её запускает
- **damageRadius** — радиус поражения
- **velocity** — скорость полёта ракеты
- **currLifeTime** – время жизни ракеты
- **currentSpeedVector** – текущий вектор скорости
- **currentCoords** – текущие координаты ракеты

Методы:

- **update_current_coords** – обновляет текущие координаты
- **update_speed_vector** – обновляет текущий вектор скорости
- **update_status** – обновляет статус объекта