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** - объект класса Диспечтер
- **gui** - UI
- **skyEnv** – Воздушная Обстановка
- **CC** - ПБУ
- **db** – база данных
- **steps** – количество шагов моделирования
- **data_colector**
- **app** - QApplication

Методы:

- **set_dispatcher()** – создаёт объект класса Диспетчер
- **set_GUI ()** – создаёт UI и возвращает данные, введённые от пользователя.
- **set_units ()** – создаёт экзепляры класса SkyEnv и CC.
- **modulate** – функция, вызывающая метод update у базовых классов в цикле по количеству шагов.
- **on_step** – возвращает количество шагов моделирования.
- **on_name** – возвращает название базы данных от пользователя.
- **on_params_ready** – вызывает у gui функцию set_session_params и сохраняет праметры моделирования.
- **run** – запускает моделирование.

Класс DatabaseManager

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

Поля:

- **db_name** - название базы данных
- **db_folder** – папка для хранения бд
- **db_path** – путь к файлу
- **cursor** - курсор
- **conn** – соединение с базой данных

Методы:

- **set_name ()** - добавление имени бд
- **create_tables ()** – создание таблиц
- **add_plane ()** – добавление самолёта
- **add_launcher()** - добавление ПУ
- **add_radar ()** - добавление радара
- **add_cc ()** - добавление ПБУ
- **load_planes ()** - считывание данных про самолёты из базы данных.
- **load_launchers ()** – прогрузка данных про ПУ из базы данных.
- **load_radars ()** – прогрузка данных про РЛС из базы данных.
- **load_cc () –** считывание данных про ПБУ из базы данных.
- **close()** – закрытие соединения с базой данных.

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.
- **logger** – объект класса Logger.
- **messageQueues** – словарь, хранящий пары: название модуля - список с сообщениями, которые отправили модулю за шаг времени.

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.

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

2. SEKilled

Поля:

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

3. SEAddRocket

Поля:

- **GUI**

- **startTime**
- **rocketId**
- **rocketCoords**

4. SEAddRocketToRadar

Поля:

- **RadarMain**
- **startTime**
- **missile**
- **rocketCoords**

5. CCLaunchMissile

Поля:

- **LauncherMain**
- **target**

6. CCToRadarNewStatus

Поля:

- **RadraMain**
- **target_new_status**

7. CCToSkyEnv

Поля:

- **SkyEnv**
- **missiles**

8. RadarToGUICurrentTarget

Поля:

- **GUI**
- **radarId**
- **targetId**
- **sectorSize**

9. RadarControllerObjects

Поля:

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

10. LaunchertoSEMissileLaunched

Поля:

- **SkyEnv**

- **targetId**
- **missile**

11. LauncherToCCMissileLaunched

Поля:

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

12. ToGuiRocketInactivated

Поля:

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

Класс Logger

Описание: обработка сообщений диспетчера и сохранение их в текстовый файл.

Поля:

- **log_dir** – директория соохранения
- **log_file** – файл для сохранения

Методы:

- **get_next_log_file** – создаёт новый log файл
- **log** – запись сообщения в файл
- **format_log_entry** – обработка сообщения для красивой записи.

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:

- **steps**
- dispatcher
- map_window
- simulation
- module_params
- on_params_save_callback
- expect_modules
- data_colector

Методы:

- init_ui
- update_params_display
- set_params_callback
- get_step
- get_db_name
- open_map_window
- show_results
- open_parameters_window
- store_parameters
- set_session_params - создание стартовой страницы для ввода данных от пользователя. Возвращает количество шагов моделирования.

Класс ParametersWindow

Поля:

- module_name
- on_save
- main_wind

Методы:

- init_ui
- create_radar_data
- create_vo_data
- create_launcher_data
- save_parameters

Класс SimulationDataCollector

Поля:

- dispatcher
- steps_data
- current_step

Методы:

- begin_step
- collect_messages
- add_message
- get_message_type

Класс MapWindow

Поля:

- simulation_steps
- current_step
- max_step
- playback_timer
- playback_speed
- text_output
- rockets
- planes
- tracked_targets
- simulation_events
- step_interv
- animation_duration
- is_playing
- plane_animation_duration
- rocket_naimation_duration
- animation_steps
- inactive_rockets
- cross_visible_time

Методы:

- setup_control_panel
- setup_step_controls
- set_simulation_data

- reset_planes_state
- play_steps
- stop_playback
- next_step
- prev_step
- update_visualization
- update_radar_targets
- update_planes
- update_zur_positions
- update_rocket_rotation
- animate_movement
- update_plane_rotation
- animate_plane_movement
- visualize_plane_track
- visualize_zur_track
- process_step
- process_message
- visualize_explosion
- remove_cross
- handle_explosion_event
- process_plane_destruction
- remove_cross_marker

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).

- **Steps** – количество шагов моделирования

- **currentStep** – шаг моделирования

Методы:

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

- **start** – инициализация всех подконтрольных юнитов.

- **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
- **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:

- **radarController** - ссылка на контроллер радаров.
- **dispatcher** - диспетчер сообщений.
- **radarId** - уникальный идентификатор радара.
- **position** - позиция радара (x,y,z).
- **maxRange** - максимальная дальность обнаружения.
- **coneAngleDeg** - угол обзора в градусах.
- **maxFollowedCount** - максимальное количество сопровождаемых целей.
- **currentTargetCount** - текущее количество сопровождаемых целей.
- **followedTargets** - словарь сопровождаемых целей {id: Target}.
- **noiseLevel** - уровень шума измерений.

Методы:

- **isTargetInRange(target, currentStep)** - проверяет, находится ли цель в зоне действия радара
- **process_target()** - обрабатывает обнаруженную цель
- **process_missile()** - обрабатывает обнаруженную ракету
- **scan(currentStep)** - выполняет цикл сканирования пространства

Класс TargetEnv

Описание: хранение объекта цели с реальными координатами.

Поля:

- **targetId** – уникальный идентификатор цели
- **clearCoords** – реальные координаты цели на каждой симуляции

Методы:

- **getCurrentCoords ()**
- **getCurrentSpeedVec ()**

Класс MissileEnv

Описание: хранение объекта ракета.

Поля:

- **missileId** - уникальный идентификатор ракеты
- **targetId** - ID цели, по которой запущена ракета
- **clearCoords** - список координат ракеты на каждом шаге

Методы:

- **getCurrentCoords ()**
- **getCurrentSpeedVec ()**

Класс RadarContoller

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

Поля:

- **allTargets** - все цели в системе {id: TargetEnv}.
- **allMissiles** - все ракеты в системе {id: MissileEnv}.
- **detectedTargets –** обнаруженные цели.

Методы:

- **addRadar** - добавляет радар в систему.
- **addDetectedTarget** – добавляет цель в список обнаруженных.
- **updateStatus** - метод, который дает конкретному радару указание отслеживать цель с заданным ID. В этом методе отсылается сообщение RadarToGUICurrentTarget.
- **processMessage** – обработка полученных сообщений
- **update** – вызывает функции обновления от всех радаров. Получение и отправка сообщений
- **killObject** – обработка уничтожения объекта
- **start** – инициализация при началае моделирования
- **addRocket** – добавление ракеты на радары
- **sendCurrentTarget** – отправляет данные о сопровождаемой цели.
- **sendDetecttedObjects** – отправляет все обнаруженные цели ПБУ.

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
- **isFollowed —** статус цели: были ли по нему выпущены ракеты.
- **attachedMissiles —** список ракет, направленных на уничтожение
- **status** – статус объекта
- **currentSpeedVector** – текущий вектор скорости
- **currentCoords** – текцщие координаты цели
- **priority** – приоритет цели

Методы:

- **attachMissile—** добавляет ракету, направленную на цель
- **detachMissile** – удаляет ракету из списка привязанных
- **updateCurrentCoords** – обновляет текущие координаты
- **updateSpeedVector** – обновляет текущий вектор скорости
- **updateStatus** – обновляет статус объекта

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
- **dispatcher** – объект класса Диспетчер
- **missile_counter** – количество ракет
- **used_ids** – использованные ракеты
- **lchr_num** – количество ПУ

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
- **get_launchers** – возвращает доступные ПУ
- **generate_missile_id** – создаёт идентификатор ракеты
- **add_launcher** – добавляет ПУ в систему
- **acknowledge** – рассылка сообщений другим модулям

Класс Launcher

Описание: Отдельная пусковая установка.

Поля:

- **silo_num —** информация о заполненности пусковых шахт.
- **ctrl** – контроллер ПУ
- **id** – уникальный идентификатор
- **coord —** координаты пусковой установки
- **silos_num** – количество ракет на установке
- **silos –** доступные ракеты
- **missile_speed_first** – скорость первого типа ракет
- **damage_radius_first** – радиус поражения первого типа ракет
- **missile_speed_second** – скорость второго типа ракет
- **damage_radius_second** – радиус поражения второго типа ракет

Методы:

- **launch** — создаёт объект Missile и добавляет его в target с помощью функции **attach_missile** . Отсылает сообщения LaunchertoSEMissileLaunched и LauncherToCCMissileLaunched
- **available_missiles** — 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