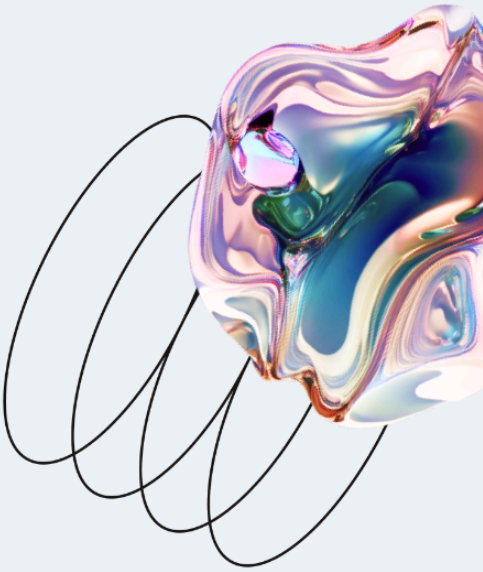
GeekBrains



Курсовая работа

Программирование на языке C. Продвинутый уровень

Оглавление

Описание проекта «Система управления сельскохозяйственного дрона»

В рамках семинаров №2, 3 и 5 вы разрабатывали проект «Система управления» на примере мини-игры «Змейка», функционал которой можно сопоставить с работой сельскохозяйственного дрона. Присутствуют схожие функции управления, сбора, выполняются задачи реального времени.

На основе данного проекта вам необходимо разработать консольное приложение – программу управления сельскохозяйственного дрона для оптимизации сбора урожая на тыквенном поле.

Требования к функционалу программы управления:

1. Дрон может перемещаться в плоскости. Перемещением дрона можно управлять вручную.
2. Дрон может определять границы тыквенного поля, эти границы ограничивают его перемещение.
3. Дрон может обнаруживать зрелые тыквы и собирать их в тележки для сбора.
4. Программа может отслеживать поведение целевых объектов:
    - появление объекта на карте – обнаружение зрелой тыквы;
    - удаление объекта с карты – зрелая тыква собрана дроном и больше не отображается на карте;
    - обновление карты – сборка урожая может происходить с некоторой периодичностью, при обновлении карты на ней появляются новые зрелые тыквы.
5. Программа отслеживает количество собранного урожая. Урожай собирается в тележки, которые за собой возит дрон. Количество прикреплённых тележек для тыкв не ограничено. При сборке тыквы длина цепочки тележек увеличивается на 1.
6. Программа дрона уведомляет пользователя об аварийной ситуации: начало цепочки тележек с собранными тыквами столкнулось с концом.
7. Дрон имеет режим автопилота: искусственный интеллект управляет дроном по заданному маршруту.
8. Сборку урожая можно проводить несколькими дронами одновременно (до пяти штук). Реализуйте кооперативный режим автопилота.

Требования к сборке приложения

- Приложение должно собираться при помощи утилиты make.
- Все прототипы функций, используемые в приложении, должны быть вынесены в отдельные файлы по функционалу, например: управление движение, ИИ, взаимодействие с объектами.
- Код программы отформатирован согласно требованиям, изученным в курсе «Программирование на языке С. Базовый уровень».

Требования к оптимизации программы:

- Произведена оптимизация по времени (скорости) работы программы.
- Произведена оптимизация производительности графики: при работе программы не допускаются задержки изображения более 100 миллисекунд, и резкие перемещения объектов на экране должны быть сведены к минимуму, не превышая 20% от общего числа кадров в секунду.

Формат сдачи работы

Работу необходимо сдать в формате отчёта:

Шаблон Отчет по курсовой работе. ФИО

Создайте копию данного шаблона и заполните:

- описание проекта – какая задача перед вами стояла и как вы её решили;
- исходный код программы – ссылка на репозиторий;
- описание файлов программы – из каких файлов состоит программа, какой файл за что отвечает и как называется;
- описание и демонстрация работы программы;
- снимки/видео/gif-анимация работы программы;
- снимок сборки программы утилитой make.

Критерии проверки работы

- Программа работает в соответствии с требованиями ТЗ каждого обязательного пункта.
- Соблюдены требования к сборке программы и форматированию кода.
- Заполнен отчёт по работе программы.

Методические рекомендации

В этом разделе приведены рекомендации, как можно использовать проект «Змейка», разработанный на семинарах №2, 3 и 5, для выполнения курсового проекта. Разберём пункты с функционалом дрона-сборщика тыкв более подробно.

Подготовка программы

В самом начале программы происходит установка начальных значений и выделение памяти:

- **snake** — голова
- **tail[]** — хвост

За это отвечают функции: **initHead(struct snake *head), initTail(struct tail_t t[], size_t size) , init(struct snake *head, struct tail *tail, size_t size).**

Голова змейки движется в соответствии с заданным направлением. Через промежуток времени **timeout(SPEED)** происходит отрисовка новой позиции головы с учётом текущего направления. Например, если направление задано как RIGHT, то это соответствует прибавлению 1 к текущей координате x **(snake.x++)**. За движение головы отвечает функция **go(struct snake *head)**.

Хвост змейки движется с учётом движения головы. За движение хвоста отвечает функция **goTail(struct snake_t *head)**.

1. Ручное управление

Коды управления змейкой и присвоенные клавиши хранятся в структурах.

Змейка управляется нажатием клавиш «вверх», «вниз», «вправо», «влево».

Для изменения направления движения используется функция **void changeDirection(snake_t* snake, const int32_t key).**

2. Сканирование и отслеживание границ перемещения

Змейка совершает **циклическое движение**, что означает невозможность выхода змейки за границы терминала.

Для обеспечения данной возможности необходимо сравнить координаты головы и максимально возможное значение координаты в текущем терминале. Для вычисления размера терминального окна используется макрос библиотеки ncurses **getmaxyx(stdscr, max_y, max_x)**.

В случае, когда координата превышает максимальное значение, происходит её обнуление.

Если координата достигает отрицательного значения, то ей присваивается соответствующее максимальное значение max_y, max_x.

3. Взаимодействие с объектами на поле

- объекты для взаимодействия – зрелые тыквы;
- взаимодействие – сбор тыквы в тележку для сбора.

В самом начале программы происходит установка начальных значений и выделение памяти food[].

За это отвечает функция: initFood(struct food f[]).

Событие сбора тыквы в тележку для сбора возникает, когда координаты головы совпадают с координатой зерна. В этом случае зерно помечается как **enable=0**.

Проверка того, является ли какое-то из зёрен съеденным, происходит при помощи функции логической функции **haveEat(struct snake *head, struct food f[ ])**: в этом случае происходит увеличение хвоста на 1 элемент**.**

В цикле `for(size_t i=0; i<MAX_FOOD_SIZE; i++)` происходит проверка наличия еды и совпадения координат `head->x == f[i].x` и `head->y == f[i].y`.

В случае выполнения условий **enable=0** и возвращается единица.

Увеличение хвоста — отвечает функция **addTail(&snake)**.

В структуре head параметр длины tsize увеличивается на единицу. `head->tsize++;`

4. Отслеживание поведения целевых объектов

- появление объекта на карте – обнаружение зрелой тыквы;
- удаление объекта с карты – зрелая тыква собрана дроном и больше не отображается на карте;
- обновление карты – сборка урожая может происходить с некоторой периодичностью, при обновлении карты на ней появляются новые зрелые тыквы.

Еда — это массив точек, состоящий из координат x, y, времени, когда данная точка была установлена, и поля, сигнализирующего, была ли данная точка съедена. Точки расставляются случайным образом в самом начале программы — **putFood(food, SEED_NUMBER), putFoodSeed(struct food *fp)**.

Если через какое-то время(FOOD_EXPIRE_SECONDS) точка устаревает, или же она была съедена(food[i].enable==0), то происходит её повторная отрисовка и обновление времени — **refreshFood(food, SEED_NUMBER).**

5. Отслеживание уровня заполнения тележки

- тележка может вместить весь урожай (не переполняется), считаем кол-во собранного урожая.

Функция **void printLevel(struct snake *head)** позволяет считать количество урожая (1 собранный урожай – 1 уровень).

6. Условия уведомления об аварийной ситуации (условия поражения змейки)

Змейка терпит поражение, когда происходит столкновение головы змейки с её хвостом.

Событие возникает, когда координаты головы совпадают с координатой хвоста. Проверка того, столкнулась ли голова с хвостом, происходит при помощи функции логической функции **isCrush(snake_t * snake)**.

В этом случае происходит автоматическое окончание игры.

При завершении игры выводится финальный результат **void printExit(struct snake *head)**

7. Режим автопилота: управление ИИ по заданному маршруту

Для реализации автопилота, путём автоматического измнениея направления к цели нужно добавить функцию `void autoChangeDirection(snake_t *snake, struct food food[], int foodSize)`. Она определяет ближайшую к себе еду и движется по направлению к ней.

8. Кооперативный режим ИИ

Вместо одной змеи, обрабатываем массив данных змей. Константа PLAYERS определяет количество игроков (змей) в программе.

Также можно сделать массив из пяти змеек с автоуправлением.

Рекомендации по улучшению проекта

По желанию вы можете улучшить свой пет-проект, добавить дополнительный функционал в программе дрона-сборщика урожая.

Дополнительный функционал работы программы дрона-сборщика урожая тыкв:

9. Сохранение и загрузка состояния системы: сборка статистики, данных, положение дрона, положение объектов
10. Условие ограниченности урожая (условие «победы»):
    - собранные тыквы не появляются заново;
    - задача собрать весь урожай тыкв на поле;
    - когда все тыквы собраны, происходит вывод сообщения “Весь урожай собран!”
    - при достижении определённого значения уровня обновление тыкв на поле больше не происходит.
11. Условие переполнения тележки:
    - возвращение на склад;
    - опустошение тележки;
    - возобновление работы.

💡 Данное задание не является обязательным к сдаче курсового проекта, но будет полезным улучшением для вашего портфолио.