Отчет по производственной практике (Case 1)

Згода, Марченко, Медынина, Пименов, Волокитин

В данном проекте реализован пошаговый бой. Основная задача — обеспечить возможность передвижения персонажей по игровому полю, выделение доступных для перемещения клеток, а также реализация механики атаки с учётом дальности и типа оружия.

Создание и отображение клеточной сетки.  
Игровое поле представляет собой двумерную сетку из клеток фиксированного размера. Каждая клетка отображается в мире Unity и может подсвечиваться разными цветами в зависимости от состояния (доступна для перемещения, атаки, подсветка при наведении мыши).

Перемещение юнитов по сетке  
Юниты могут перемещаться по сетке с ограничением по количеству доступных очков движения. При выборе клетки, на которую юнит может пройти, она подсвечивается, и юнит плавно перемещается по рассчитанному пути.

Подсветка доступных клеток для перемещения и атаки  
Система подсвечивает клетки, до которых юнит может дойти за оставшиеся очки движения, а также клетки, находящиеся в пределах радиуса атаки с учётом типа атаки (ближний бой, стрельба, магия).

Механика атаки с проверкой дистанции  
Юнит может выполнять атаки разного типа, при этом проверяется, находится ли цель в пределах допустимой дальности. Если цель вне досягаемости, атака не выполняется, и ход игрока не завершается, что позволяет выбрать другую цель или переместиться.

Управление состояниями юнита  
Ведётся учёт очков движения и действий юнита за ход, состояния живости и смерти, а также обработка завершения ходов и смены состояний.

Конец игры  
Игра завершается либо когда юниты одной из команды мертвы, либо когда счетчик ходов будет равен 10. Тогда победа определяется наибольшим суммарным здоровьем одной из команд.

Grid.cs

Этот файл реализует класс, отвечающий за создание и управление игровой клеточной сеткой.

* **Создание сетки**: Генерируется двумерный массив клеток заданного размера. Каждая клетка создаётся как игровой объект и располагается в пространстве с определённым шагом, соответствующим размеру клетки.
* **Получение информации о клетках**: Реализованы методы для получения конкретной клетки по её координатам, а также проверки доступности клетки для передвижения (например, не занята ли она другим объектом).
* **Подсветка клеток**: Класс умеет подсвечивать клетки на игровом поле, которые доступны для перемещения или атаки. При этом выделяются только те клетки, которые соответствуют заданным ограничениям (например, в пределах очков движения или радиуса атаки).
* **Определение доступных для перемещения клеток**: С помощью обхода в ширину (BFS) и проверки проходимости клеток вычисляются все клетки, куда юнит может дойти, учитывая текущие очки движения и препятствия на пути.
* **Определение клеток в радиусе атаки**: Вычисляются все клетки, находящиеся в пределах радиуса огневой атаки юнита (например, радиус стрельбы), без учёта препятствий на пути.
* **Управление подсветкой**: Есть возможность очистить все подсветки на клетках, чтобы переключаться между режимами отображения (перемещение, атака и т.д.).

UnitSpawner.cs

Этот файл содержит класс, отвечающий за создание и размещение игровых юнитов на клеточной сетке.

* **Связь с сеткой**: Класс получает ссылку на игровую сетку, чтобы корректно рассчитывать позицию появления юнитов на поле.
* **Создание юнитов**: Метод отвечает за спавн (создание) юнита на заданных координатах сетки. При этом создаётся игровой объект на основе префаба, позиционируется в нужной точке с учётом размера клетки.
* **Инициализация свойств юнита**: После создания, юниту назначаются базовые характеристики, такие как здоровье, фракция, урон, дальность атаки ближнего и дальнего боя.
* **Визуальное оформление**: В зависимости от принадлежности юнита к игроку или врагу, меняется цвет материала для быстрой визуальной идентификации.
* **Регистрация юнита**: Созданный юнит регистрируется в менеджере юнитов для дальнейшего управления и взаимодействия в игровом процессе.
* **Массовый спавн**: Реализован метод для одновременного создания нескольких юнитов с заранее заданными координатами и принадлежностью к фракциям, что позволяет быстро расставить начальные позиции юнитов на поле.

Unit.cs

Этот файл содержит класс, описывающий игрового юнита — отдельную боевую единицу на игровом поле.

* **Позиционирование на сетке**: Юнит хранит свои координаты на клеточной сетке и может перемещаться по ней с заданной скоростью.
* **Движение**: Юнит принимает путь (набор клеток), по которому последовательно перемещается, используя очередь позиций для плавного перемещения по миру.
* **Характеристики и состояние**: У юнита есть здоровье, параметры атаки (тип, урон, дальность для разных видов атаки), а также показатели хода (максимальное и текущее количество очков движения).
* **Атаки**: Юнит умеет выполнять атаки разного типа — ближний бой, стрельба и магия, проверяя при этом, находится ли цель в пределах соответствующей дальности. Если цель вне досягаемости, атака не совершается, и юнит возвращается к выбору действий.
* **Получение урона и смерть**: Юнит может получать урон, уменьшать здоровье и умирать, при этом уведомляя менеджер юнитов и менеджер ходов об уничтожении.
* **Обновление интерфейса**: Отображает здоровье в интерфейсе, автоматически обновляя текст при изменениях.
* **Управление ходом**: Отслеживает, совершил ли юнит движение и атаку за текущий ход, и умеет сбрасывать эти состояния для следующего хода.
* **События**: Использует делегат OnActionComplete для уведомления системы о завершении движения или атаки, что позволяет синхронизировать игровой процесс.

GridCellData.cs

Этот файл содержит класс, который представляет отдельную ячейку (клетку) игровой сетки.

* **Координаты клетки**: Хранит внутренние координаты клетки по осям X и Z на сетке.
* **Состояние занятости**: Отвечает за информацию, занята ли данная клетка каким-либо объектом (например, юнитом), чтобы нельзя было ходить по уже занятой клетке.
* **Связь с игровым объектом**: Содержит ссылку на объект Unity, который визуально представляет клетку на сцене (например, сама игровая плитка).

Pathfinding.cs

Этот файл реализует систему поиска пути по игровому полю с использованием алгоритма A\*.

* **Поиск пути:** Метод FindPath принимает начальные и конечные координаты и вычисляет оптимальный путь между этими точками, учитывая проходимость клеток.
* **Алгоритм A\*:** Для определения кратчайшего пути используется эвристика (расстояние Манхэттена), а также расчет стоимости движения по клеткам. Алгоритм поддерживает открытый список для узлов, которые нужно исследовать, и закрытый набор уже проверенных позиций.
* **Восстановление пути:** После нахождения целевой клетки путь восстанавливается, формируя список координат клеток от начальной до конечной.
* **Поддержка направлений:** Обработка соседних клеток происходит в четырёх основных направлениях: вверх, вниз, влево и вправо.
* **Singleton:** Класс имеет статическое свойство Instance для удобного глобального доступа к функционалу поиска пути.

PathNode.cs

Данный файл содержит класс PathNode, который представляет собой узел в алгоритме поиска пути (A\*).

* **Координаты:** Хранит координаты узла на сетке (X и Z).
* **Стоимость перемещения:** Свойства GCost (стоимость от начальной точки до текущей), HCost (эвристическая оценка оставшегося пути до цели) и FCost (сумма GCost и HCost), которые используются для оценки оптимальности пути.
* **Родительский узел:** Свойство Parent хранит ссылку на предыдущий узел в найденном пути, что позволяет восстановить полный маршрут после завершения поиска.
* **Позиция:** Свойство Pos возвращает координаты узла в виде Vector2Int для удобства использования в алгоритме.

UnitManager.cs

Этот файл реализует класс UnitManager, который отвечает за управление всеми игровыми юнитами на поле.

* **Синглтон:** Класс реализован как синглтон, чтобы обеспечить единую точку доступа к управлению юнитами.
* **Хранение юнитов:** Ведёт отдельные списки для юнитов игрока (PlayerUnits) и врагов (EnemyUnits).
* **Регистрация и удаление:** Позволяет регистрировать новых юнитов и удалять погибших, поддерживая актуальные списки.
* **Получение всех юнитов:** Предоставляет возможность получить список всех активных юнитов на поле.
* **Поиск ближайшего юнита игрока:** Имеется функция, которая для заданного юнита врага ищет ближайший живой юнит игрока по расстоянию на сетке.
* **Проверка занятости клетки:** Позволяет проверить, занят ли определённый клеточный координат юнитом.
* **Определение победителя:** Рассчитывает сумму здоровья юнитов обеих сторон и определяет, кто победил, или фиксирует ничью.

BattleUIManager.cs

Этот скрипт отвечает за управление элементами пользовательского интерфейса (UI) в боевой сцене.

* **Отображение информации о ходе:** Обновляет текст на экране, показывая текущий номер хода, когда начинается новый раунд.
* **Отображение результата игры:** Показывает текст с информацией о победителе в конце игры.
* **Выбор типа атаки:** Позволяет игроку переключать тип атаки выбранного юнита между ближним боем, стрельбой и магией. При выборе стрельбы дополнительно подсвечиваются клетки, доступные для атаки.
* **Взаимодействие с другими системами:** Подписывается на события менеджера ходов (TurnManager), чтобы обновлять UI в нужные моменты (начало раунда, завершение игры).

EnemyAI.cs

Этот скрипт реализует логику искусственного интеллекта (ИИ) для вражеских юнитов в игре.

* **Управление ходом врага:** Когда наступает очередь врага, запускается корутина, которая с небольшой задержкой выполняет действия.
* **Поиск цели:** ИИ ищет ближайший юнит игрока, чтобы либо атаковать его, либо приблизиться к нему.
* **Атака:** Если цель находится в пределах ближнего или дальнего боя, враг совершает соответствующую атаку (ближний бой или стрельба).
* **Передвижение:** Если цель слишком далеко для атаки, ИИ ищет ближайшую свободную клетку рядом с целью и планирует путь к ней, после чего перемещается по этому пути.
* **Обработка завершения хода:** По окончании действий врага вызывается сигнал о завершении его хода в общем менеджере ходов.

TurnManager.cs

Этот скрипт управляет логикой пошаговой системы игры, контролируя порядок ходов юнитов и смену раундов.

* **Singleton:** Класс реализован как синглтон (Instance), чтобы обеспечить единственный центральный менеджер ходов.
* **Очередь ходов:** Все живые юниты в начале раунда добавляются в очередь, которая определяет порядок их ходов.
* **Раунды:** Поддерживается счётчик текущего раунда (\_currentRound), с событием RoundStarted, уведомляющим UI или другие системы о начале нового раунда.
* **Переход между ходами:** Метод StartNextTurn() выбирает следующего живого юнита из очереди и запускает его ход:
* Если это юнит игрока — передаёт управление системе PlayerInput.
* Если враг — запускает ИИ.
* **Завершение хода:** Когда текущий юнит заканчивает ход, вызывается EndCurrentTurn(), который либо начинает ход следующего юнита, либо начинает новый раунд при окончании всех ходов.
* **Обработка смерти:** Если юнит умирает во время или до своего хода, его ход корректно завершается или удаляется из очереди.
* **Завершение игры:** После 10 раундов вызывается событие окончания игры, деактивируются ввод игрока и ИИ, и выводится победитель.

Cell.cs

Этот скрипт отвечает за визуальное выделение отдельной клетки игрового поля.

* **Отрисовка и цвета:**  
  Используется компонент Renderer для управления цветом контура клетки через параметр шейдера \_OutColor.  
  Хранятся цвета для разных состояний:
  + hoverColor — цвет при наведении курсора (жёлтый).
  + moveHighlightColor — цвет при выделении клетки, куда можно пойти (голубой).
  + attackHighlightColor — цвет при выделении клетки в зоне атаки (красный).
* **Обработка мыши:**  
  При наведении мыши (OnMouseEnter) если сейчас ход игрока, меняется цвет контура на hoverColor.  
  При уходе мыши с клетки (OnMouseExit) цвет возвращается либо к текущему выделению (если есть), либо к исходному цвету.
* **Методы управления выделением:**
  + HighlightMove() — выделить клетку цветом ходов (голубой).
  + HighlightShootAttack() — выделить клетку цветом зоны атаки (красный).
  + ClearHighlight() — снять выделение и вернуть исходный цвет.
* **Логика выделения:**  
  Переменная isHighlighted и currentHighlightColor отслеживают, выделена ли клетка и каким цветом, чтобы корректно управлять сменой цветов при наведении и снятии курсора.

PlayerInput.cs

* **Singleton:**  
  Используется паттерн одиночки через публичное статическое свойство Instance для удобного доступа из других частей игры.
* **Управление юнитом:**  
  Метод StartPlayerTurn(Unit unit) вызывается для передачи контроля над юнитом игроку, устанавливая внутреннюю переменную \_controlledUnit.
* **Обработка ввода мыши в методе** Update**:**

Indicator.cs

Класс Indicator отвечает за визуальный индикатор, который следует за определённым объектом в игровом мире.

* **Singleton:**  
  Класс реализован как синглтон через публичное статическое свойство Instance, доступное во всём проекте.
* **Поле** \_following**:**  
  Хранит Transform объекта, за которым должен следовать индикатор.
* **Метод** Follow(GameObject following)**:**  
  Позволяет задать объект, за которым индикатор будет следить. Сохраняет его трансформ.
* **Метод** LateUpdate()**:**  
  Каждый кадр после всех обновлений проверяет, установлен ли объект для слежения. Если да — устанавливает позицию индикатора чуть выше позиции объекта (смещение по Y на 1 единицу).  
  Это гарантирует, что индикатор всегда отображается над объектом.
* **Метод** GetFollowingPointPosition()**:**  
  Возвращает позицию, на которую должен быть установлен индикатор — позиция объекта + смещение по высоте.

Projectile.cs

Класс Projectile представляет собой снаряд, который движется в сторону цели и наносит урон при достижении.

* **Метод** Initialize(Unit target, int damage)**:**  
  Инициализирует снаряд, устанавливая цель и количество наносимого урона.
* **Метод** Update()**:**  
  Каждый кадр:
  + Проверяет, существует ли цель и жива ли она. Если нет — уничтожает снаряд.
  + Двигает снаряд в направлении цели с заданной скоростью.
  + Если снаряд приблизился к цели ближе, чем на 0.3 единицы, наносит урон цели и уничтожает себя.

CompositionRoot.cs

**Класс** CompositionRoot **— это точка инициализации и связывания основных компонентов игры.**

* Его задача — создать и настроить ключевые объекты и сервисы, необходимые для работы игрового процесса.
* В методе Awake он инициализирует объекты: сетку (Grid), спаунер юнитов (UnitSpawner), систему поиска пути (Pathfinding) и менеджер юнитов (UnitManager).
* После создания и настройки этих компонентов запускается генерация игрового поля и размещение юнитов на нем.
* Таким образом, CompositionRoot отвечает за начальную подготовку и «сборку» всех основных систем в единую рабочую структуру.







