Для создания модели приложения, в основе которой лежит пошаговая тактическая игра, необходимо разработать основные структуры данных и алгоритмы, которые будут использоваться для решения задач игры, описанных в техническом задании.

**1. Структуры данных**

Основные структуры данных, используемые в приложении, включают:

* **Игровое поле**: Модель игрового поля будет представлена в виде двумерной матрицы (списка списков), где каждая клетка может быть пустой, занята персонажем игрока или персонажем противника, либо же препятствием.

Например:

игровое\_поле = [

[0, 0, 0, 'X', 0, 0, 0, 0],

[0, 'player1', 0, 0, 'enemy1', 0, 0, 0],

[0, 0, 'X', 0, 0, 'X', 0, 0],

[0, 0, 0, 'player2', 0, 0, 0, 'enemy2'],

[0, 0, 0, 0, 0, 0, 0, 0],

[0, 'X', 0, 0, 0, 0, 'player3', 0],

[0, 0, 0, 0, 0, 'enemy3', 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0],

]

Где:

* + **0** – пустая клетка,
  + **'playerX'** и **'enemyX'** – персонажи игрока и противника,
  + **'X'** – препятствие (непроходимая клетка).
* **Персонажи**: Каждый персонаж (игрок или противник) представлен объектом класса Character, который содержит атрибуты здоровья, урона, типа атаки, радиуса атаки и координат.

Например:

class Character:

def \_\_init\_\_(self, name, hp, damage, attack\_type, attack\_range, position):

self.name = name

self.hp = hp

self.damage = damage

self.attack\_type = attack\_type # "Ближний бой" или "Дальний бой"

self.attack\_range = attack\_range # радиус атаки в клетках

self.position = position # (x, y) координаты

* **Объекты игры**: Все персонажи (игроки и противники) будут храниться в отдельных списках для удобства взаимодействия:

игроки = [Character(...), Character(...)]

противники = [Character(...), Character(...)]

**2. Алгоритмы**

Для реализации игровых механик потребуются следующие ключевые алгоритмы:

**2.1 Алгоритм перемещения персонажей**

Каждый персонаж может перемещаться на определенное количество клеток (например, не более 3 клеток за ход). Для перемещения необходимо:

* Проверить, доступна ли клетка для перемещения (не занята другим персонажем или препятствием).
* Изменить координаты персонажа на игровом поле.

Пример:

def переместить\_персонажа(character, new\_position, игровое\_поле):

x, y = new\_position

if игровое\_поле[x][y] == 0: # клетка пустая

игровое\_поле[character.position[0]][character.position[1]] = 0 # освободить старую клетку

character.position = new\_position

игровое\_поле[x][y] = character.name # занять новую клетку

**2.2 Алгоритм атаки**

Для атаки персонаж должен находиться в радиусе атаки от врага. Если враг попадает в радиус атаки, ему наносится урон.

Пример:

def атаковать(character, target, игровое\_поле):

if расстояние(character.position, target.position) <= character.attack\_range:

target.hp -= character.damage

if target.hp <= 0:

игровое\_поле[target.position[0]][target.position[1]] = 0 # убрать персонажа с поля

**2.3 Алгоритм действий противника**

Для искусственного интеллекта необходимо реализовать алгоритм принятия решений, который может включать:

* Перемещение ближе к игроку, если персонаж противника находится вне радиуса атаки.
* Атака ближайшего персонажа игрока, если он находится в радиусе.

Пример:

def ход\_AI(противник, игроки, игровое\_поле):

ближайший\_игрок = найти\_ближайшего(противник, игроки)

if расстояние(противник.position, ближайший\_игрок.position) <= противник.attack\_range:

атаковать(противник, ближайший\_игрок, игровое\_поле)

else:

переместиться\_ближе(противник, ближайший\_игрок, игровое\_поле)

**2.4 Алгоритм проверки окончания игры**

Игра заканчивается, если все персонажи одной из сторон уничтожены.

Пример:

def проверка\_окончания\_игры(игроки, противники):

if all(player.hp <= 0 for player in игроки):

return "Победа противника"

elif all(enemy.hp <= 0 for enemy in противники):

return "Победа игрока"

return None

**3. Дополнительные элементы модели**

* **Контроль корректности ходов**: Проверка допустимости перемещения и атаки для каждого хода, чтобы предотвратить ошибки (например, попытку атаки из недопустимого радиуса).
* **Отображение информации**: Обновление панели информации для отображения характеристик выбранного персонажа.

**4. Итеративность проектирования**

Проектирование игры основывается на итеративном процессе, где возможны уточнения выбранных структур данных и алгоритмов на следующих этапах разработки. В ходе тестирования будут выявляться возможные улучшения в механике игры или в оптимизации кода.