1. ВВЕДЕНИЕ

Предлагаемый пример проектной документации демонстрирует результат применения автоматного подхода (SWITCH-технологии) к созданию игрового приложения "Type & Slayer". Рассматриваемое программное обеспечение разработано как учебный пример, иллюстрирующий, как конечные автоматы могут эффективно описывать логику игрового процесса, управление состояниями игры и поведение персонажей.

Разработанная игра состоит из следующих ключевых частей:

Ядро игры (Game) – управляет основным игровым циклом, обрабатывает события и координирует работу всех подсистем. Его поведение описывается автоматом состояний игры (A0).

Сущности (Player, Enemy) – моделируют игрока и врагов. Поведение врага описывается автоматом поведения врага (A1).

Подсистема ввода (VirtualKeyboard) – обеспечивает визуализацию и обратную связь при вводе текста.

Генератор контента (WordGenerator) – предоставляет слова для игры в зависимости от уровня сложности.

Отметим, что данный пример намеренно упрощен для демонстрации автоматного подхода: автоматы не являются вложенными и обмениваются информацией через общие данные (состояние игрока, список врагов).

2. ЧАСТНОЕ ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА ИГРУ "TYPE & SLAYER"

1. ВВЕДЕНИЕ

Наименование разработки: “Игра 'Type & Slayer'” (далее "игра").
Игра предназначена для изучения основ автоматного программирования на примере игрового приложения.

2. НАЗНАЧЕНИЕ РАЗРАБОТКИ

Игра предназначена для демонстрации принципов конечных автоматов в игровой механике. Игровой процесс сочетает в себе элементы жанра "рогалик" (Roguelike) и клавиатурного тренажера.

3. ТРЕБОВАНИЯ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ

3.1. Требования к функциональным характеристикам
3.1.1. Требования к инструментальной среде

3.1.1.1. Игра предназначена для функционирования под управлением ОС семейства Windows, Linux или macOS.
3.1.1.2. Разработка игры выполняется с использованием интерпретатора Python версии 3.9+ и библиотеки Pygame.

3.1.2. Требования к пользовательскому интерфейсу
3.1.2.1. Игра работает в графическом окне размером не менее 1024x768 пикселей.
3.1.2.2. Интерфейс должен содержать следующие элементы:
* Игровое поле, на котором отображаются игрок (слева) и враги (справа).
* Визуальную клавиатуру в нижней части экрана для обратной связи при вводе.
* Поле ввода текста.
* Панель состояния игрока (здоровье, опыт, счет, уровень).
3.1.2.3. Враги, слово на которых совпадает с текущим вводом игрока, должны подсвечиваться.
3.1.2.4. При проигрыше или победе должны отображаться соответствующие информационные экраны с возможностью начать игру заново (клавиша R).

3.1.3. Требования к реализуемым функциям
3.1.3.1. Игровой процесс:
* Игрок управляется косвенно, путем набора слов на врагах.
* Враги появляются справа и движутся влево к игроку.
* Цель игрока — уничтожать врагов, набирая соответствующие слова, прежде чем они коснутся его.
3.1.3.2. Система ввода:
* Игрок вводит слово с физической клавиатуры.
* Введенный текст отображается в поле ввода.
* Клавиша Backspace удаляет последний символ.
* Клавиша Enter подтверждает ввод и запускает проверку слова.
* При совпадении введенного слова со словом на враге, враг уничтожается, начисляются очки и опыт.
* Визуальная клавиатура подсвечивает нажатые клавиши.
3.1.3.3. Система прогрессии:
* У игрока есть здоровье, которое уменьшается при касании врага.
* У игрока есть опыт. Получение опыта повышает уровень игрока.
* Повышение уровня увеличивает максимальное здоровье и сложность игры (враги появляются чаще).
3.1.3.4. Генерация контента:
* Слова для врагов выбираются из предопределенного списка в зависимости от текущего уровня игрока.
* Скорость движения врагов зависит от длины слова.

3.2. Требования к надежности
3.2.1. Должна быть обеспечена корректная обработка ввода, не приводящая к "залипанию" клавиш или неверному определению введенных слов.
3.2.2. Игра должна корректно завершаться и освобождать ресурсы (экраны, таймеры) при закрытии окна.

3.3. Требования к составу и параметрам технических средств
3.3.1. Особые требования к производительности и объему памяти не предъявляются.

3.4. Требования к информационной и программной совместимости
3.4.1. Обмен информацией между подсистемами осуществляется через вызовы методов объектов и чтение/запись их атрибутов (например, player.health, enemies).

3.5. Специальные требования
3.5.1. Поведение игры должно быть описано с помощью конечных автоматов. Состояния игры: "Активная игра", "Поражение", "Победа". Состояния врага: "Движение", "Атака" (касание игрока).

4. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

По окончанию проектирования Разработчик предоставляет Заказчику следующие документы:

Перечни событий, входных переменных и выходных воздействий, участвующих в работе игры.

Схемы связей и графы переходов конечных автоматов, специфицирующих поведение игры.

Пример протокола работы игры для различных сценариев.

3. СТРУКТУРНАЯ СХЕМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
3.1. Общая структурная схема программы
(Здесь должна быть схема, показывающая взаимосвязь main.py, game.py (автомат A0), enemy.py (автомат A1) и вспомогательных модулей.)

3.2. Порядок взаимодействия частей подсистемы
Основной цикл в main.py получает события от пользователя и время, передавая их в метод handle_event() и update() объекта Game. Game обновляет состояние врагов, проверяет условия перехода для автомата A0 (игры) и, в свою очередь, обновляет

состояние каждого врага (автомат A1). Визуальная клавиатура обновляется и отрисовывается на основе тех же событий ввода.

4. ПЕРЕЧЕНЬ И НУМЕРАЦИЯ СОБЫТИЙ
Для автомата состояний игры A0

Номер события Описание
10 Игрок ввел слово и нажал Enter (e_check_word).
20 Здоровье игрока достигло нуля (e_player_died).
30 Условия победы выполнены (не реализовано в текущей версии, зарезервировано) (e_victory).
40 Нажата клавиша 'R' на экране поражения/победы (e_restart).
Для автомата поведения врага A1

Номер события Описание
50 Произошел "тактовый" импульс времени (e_tick).
60 Враг коснулся границы области игрока (e_reached_player).
70 Слово врага совпало с введенным игроком (e_word_matched).

5. ПЕРЕЧЕНЬ И НУМЕРАЦИЯ ВХОДНЫХ ПЕРЕМЕННЫХ

Для автомата состояний игры A0

Переменная Описание
x10 Здоровье игрока (player.health == 0).
x20 Условие победы (зарезервировано, всегда False).
Для автомата поведения врага A1

Переменная Описание
x50 Флаг того, что враг коснулся левого края (enemy.x <= 50).
x60 Флаг совпадения слова врага с глобальным typed_word.
x70 Флаг того, что враг еще жив и не уничтожен (всегда True до уничтожения).

6. ПЕРЕЧЕНЬ И НУМЕРАЦИЯ ВЫХОДНЫХ ВОЗДЕЙСТВИЙ

Для автомата состояний игры A0

Воздействие Описание
z10 Сменить режим отрисовки на "Поражение" (set_game_over).
z20 Сменить режим отрисовки на "Победа" (set_victory).

z30 Сбросить все игровые параметры до начальных (reset_game).
z40 Проверить введенное слово и нанести урон (check_word_and_damage).
Для автомата поведения врага A1

Воздействие Описание
z50 Нанести урон игроку (player.take_damage).
z60 Увеличить счет и опыт игрока (player.gain_exp, game.score++).
z70 Уничтожить объект врага (remove from game.enemies).

7. ПОЯСНЕНИЯ К ИСПОЛЬЗУЕМОЙ НОТАЦИИ

(Аналогично разделу 7 из PDF-примера)

An - автомат с номером n.

yn - переменная состояния автомата с номером n.

xj - входная переменная с номером j.

zk - выходное воздействие с номером k.

en - событие с номером n. В условиях переходов "en" является сокращенной записью предиката "e == n".

8. СИСТЕМОНЕЗАВИСИМАЯ ЧАСТЬ

8.1. Автомат состояний игры (A0)

8.1.1. Словесное описание

Автомат A0 управляет глобальным состоянием игрового процесса. Он запускается при старте новой игры и реагирует на ключевые игровые события: проверку введенного слова, смерть игрока и команду рестарта.

В состоянии Активная игра (y0=0) происходит нормальный игровой процесс. При нажатии Enter (событие e10) автомат инициирует проверку введенного слова (воздействие z40). Если здоровье игрока падает до нуля (переменная x10 истинна),

автомат переходит в состояние Поражение (y0=1) и активирует соответствующий экран (воздействие z10).

В состоянии Поражение (y0=1) игровой процесс заморожен. При нажатии клавиши R (событие e40) автомат переходит обратно в состояние Активная игра, выполняя сброс всех параметров игры (воздействие z30).

8.1.2. Схема связей и граф переходов

Автомат состояний игры. Схема связей автомата

[События: e10, e40] --> [Автомат A0] --> [Воздействия: z40, z10, z30]
[Входные переменные: x10] --^

Автомат состояний игры. Граф переходов

(Описание графа)

Состояние 0 (Активная игра):

При e10: / z40. (Остаться в состоянии 0).

При x10: / z10. -> Состояние 1.

Состояние 1 (Поражение):

При e40: / z30. -> Состояние 0.

8.1.3. Текст функции, реализующей автомат (псевдокод)

```
# Автомат состояний игры A0

import pygame
import random
from entities.player import Player
from entities.enemy import Enemy
from entities.keyboard import VirtualKeyboard
from utils.word_generator import WordGenerator

class Game:
    def __init__(self, screen, width, height):
```

```
self.screen = screen
self.width = width
self.height = height
self.clock = pygame.time.Clock()

# Цвета
self.colors = {
    'background': (20, 20, 30),
    'text': (255, 255, 255),
    'enemy': (255, 100, 100),
    'enemy_hover': (255, 150, 150),
    'ui': (50, 50, 70),
    'health': (255, 80, 80),
    'exp': (100, 255, 100),
    'keyboard_bg': (30, 30, 40),
    'key_normal': (60, 60, 80),
    'key_pressed': (100, 100, 150),
    'key_special': (80, 80, 120),
}

# Игровые объекты
self.player = Player(50, height // 2)
self.word_generator = WordGenerator()
self.enemies = []
self.keyboard = VirtualKeyboard(50, height - 220, self.colors)

# Игровые параметры
self.score = 0
self.level = 1
self.enemy_spawn_timer = 0
self.spawn_delay = 2.0  # Секунды между спавном
self.current_word = ""
self.typed_word = ""

# Шрифты
pygame.font.init()
self.title_font = pygame.font.Font(None, 48)
self.main_font = pygame.font.Font(None, 36)
self.small_font = pygame.font.Font(None, 24)
```

```
    # Статус игры
    self.game_over = False
    self.victory = False

def handle_event(self, event):
    if self.game_over or self.victory:
        if event.type == pygame.KEYDOWN and event.key == pygame.K_r:
            self.reset_game()
        return

    if event.type == pygame.KEYDOWN:
        # Обработка ввода
        if event.key == pygame.K_BACKSPACE:
            self.typed_word = self.typed_word[:-1]
            self.keyboard.press_key('backspace')
        elif event.key == pygame.K_RETURN:
            self.check_word()
            self.keyboard.press_key('enter')
        elif event.key == pygame.K_ESCAPE:
            self.typed_word = ""
        else:
            # Добавляем символ, если это буква
            char = event.unicode.lower()
            if char.isalpha():
                self.typed_word += char
                self.keyboard.press_key(char)

def check_word(self):
    """Проверка введенного слова"""
    for enemy in self.enemies[:]:
        if enemy.word == self.typed_word:
            self.enemies.remove(enemy)
            self.score += enemy.word_length * 10
            self.player.gain_exp(10)
            self.typed_word = ""
            return
    # Если слово не найдено
    self.typed_word = ""

def spawn_enemy(self):
```

```
        """Создание нового врага"""
        if len(self.enemies) < 5 + self.level * 2:
            word = self.word_generator.get_random_word(self.level)
            x = self.width
            y = random.randint(100, self.height - 300)
            enemy = Enemy(word, x, y, self.colors)
            self.enemies.append(enemy)

    def update(self, dt):
        if self.game_over or self.victory:
            return

        # Обновление врагов
        for enemy in self.enemies[:]:
            enemy.update(dt)
            # Проверка достижения левого края
            if enemy.x < 50:
                self.player.take_damage(enemy.word_length)
                self.enemies.remove(enemy)

        # Проверка здоровья игрока
        if self.player.health <= 0:
            self.game_over = True

        # Проверка уровня
        if self.player.exp >= self.player.exp_to_next:
            self.level_up()

        # Спавн врагов
        self.enemy_spawn_timer += dt
        if self.enemy_spawn_timer >= self.spawn_delay:
            self.spawn_enemy()
            self.enemy_spawn_timer = 0
            self.spawn_delay = max(0.5, 2.0 - self.level * 0.2)

        # Обновление клавиатуры
            self.keyboard.update(dt)

    def level_up(self):
        """Повышение уровня"""
```

```
self.level += 1
self.player.level_up()
```