Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **КОСМИЧЕСКИЙ**

КАФЕДРА **КЗ «Прикладная математика, информатика и вычислительная техника»**

ОТЧЕТ ПО КУРСОВОЙ РАБОТЕ

Дисциплина: Программирование на ЯВУ

Название: Компьютерная игра Tomb Raider: The Origins

Студент КЗ-34Б (Группа) Р. Д. Павлов (И. О. Фамилия)

Преподаватель А. В. Афанасьев (И. О. Фамилия)

Мытищи, 2023

Содержание

Введение

«Tomb Raider: The Origins» – компьютерная 2D игра с видом сверху, которая поддерживается на ОС Windows.

Главный герой появляется в гробнице на стартовой локации со входом, который также является выходом для прохождения игры. Помимо этой локации существуют еще 25 комнат, с заранее созданными интерьерами (11 видов комнат), которые включают врагов, ловушки (шипы), стены, двери, ведущие к другим комнатам, различные виды сундуков с определенными наградами.

Чтобы пройти игру, нужно добраться до комнаты с голубым сундуком, в котором находится сокровище, вернуться в начальную локацию и выйти из гробницы.

Чтобы одолеть противников в игре есть различные виды оружия, снаряжения, и расходников, которые имеют разные характеристики, что добавляет определенную составляющую геймплея, заключающуюся в поиске более ценных предметов для простого прохождения игры.

Например, на старте игроку выдается пистолет, с которым проблематично убивать врагов, или, например, автомат Калашникова, который игрок может найти в сундуке или найти при убийстве врагов. Или же зелья восстановления здоровья или баффа (сброса всех негативных эффектов замедления и временного ускорения).

В игре присутствуют все необходимые визуальные составляющие, такие как меню различных состояний игры (инвентарь/пауза, начальное меню, экран смерти и т.д.), интерфейсы состояния главного героя и радара, спрайты существ и предметов.

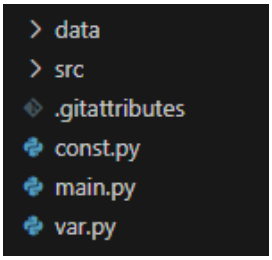
Режим игры: Одиночная

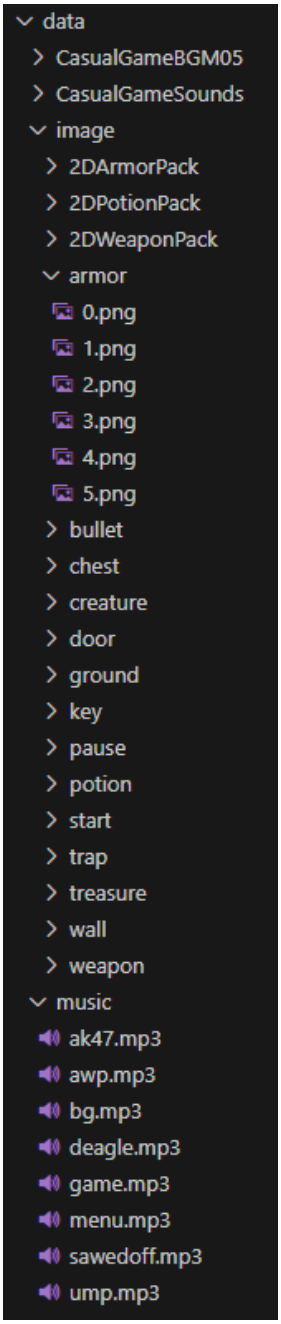
Жанр игры: Адвенчура (приключенческая игра)

Алгоритм программы

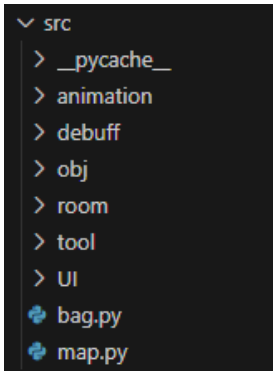
Для написания программы я использовал ЯП Python (3.11.1), работал в среде разработки Visual Studio Code и использовал «игровую библиотеку» Pygame (набор инструментов, который включает в себя возможность взаимодействовать или создавать: графику и анимацию, звук (включая музыку), управление (мышь, клавиатура).

Программа использует множество файлов, которые представляют собой спрайты (картинки-модельки), а также звуки и музыку. Они хранятся в папке data.





В папке src находятся только файлы .py, которые представляют собой различные классы и расположены они в подпапках с соответствующими именами (кроме bag и map).



В дальнейшем описании программы в представленных блоках кода добавлены комментарии, которые дают понять, что происходит в нижерасположенной части кода, что сэкономит время и сделает более конкретным описание алгоритма.

main.py

```
import pygame
import const
import var
from src.UI.interface_start import START
from src.UI.interface_play import PLAY
from src.tool.vector import Vector


if __name__ == '__main__':
    # Инициализация pygame
    pygame.init()
    pygame.display.set_caption('Tomb Raider: The Origins')
    # Инициализация файла const, включающего звуки, изображения и анимации
    const.Init()
    # Инициализация START и PLAY интерфейсов и установка START активным
    var.interface = var.start = START()
    var.play = PLAY()

    # Предоставление информации о частоте кадров
    clock = pygame.time.Clock()

    while not var.quit:
        clock.tick(const.FPS)
        # Обновляем ввод с клавиатуры и мыши
        var.key_up.clear()
        var.key_down.clear()
        var.mouse_up.clear()
        var.mouse_down.clear()
        var.mouse = Vector(pygame.mouse.get_pos()[0], pygame.mouse.get_pos()[1])

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                var.quit = True
            elif event.type == pygame.KEYUP:
                var.key_up += [event]
            elif event.type == pygame.KEYDOWN:
                var.key_down += [event]
            elif event.type == pygame.MOUSEBUTTONUP:
                var.mouse_up += [event]
            elif event.type == pygame.MOUSEBUTTONDOWN:
                var.mouse_down += [event]
        # Обновляем интерфейс и задаем задний фон экрана
        var.interface.update()
        var.screen.fill((0, 0, 0))
        var.interface.draw(var.screen)
        # Отображаем частоту кадров в левом верхнем углу
        var.screen.blit(pygame.font.Font(None, 30).render("FPS: %d" % clock.get_fps(), True,
     (255, 255, 255)), (20, 10))
        pygame.display.update()
    pygame.quit()
```

const.py

```
import pygame
import os
import var
from src.animation.animation_repository import AnimationRepository

"""
Файл содержащий постоянные объект или переменные
"""


# Размер экрана в пикселях.
SCREEN_SIZE = (1920, 1080)
# Комната состоит из множества плиток. Это размер одной плитки в пикселях.
TILE_SIZE = (35, 35)
# Комната состоит из плиток 20*20 по ширине и высоте.
ROOM_SIZE = (int(SCREEN_SIZE[0]/60) - 1, int(SCREEN_SIZE[0]/60) - 2)
# Карта состоит максимум из комнат 6*6 по ширине и высоте.
MAP_SIZE = (4, 4)
FPS = 60
MAX_ENEMY = 10
MIN_ENEMY = 1
# Используется для дерева квадрантов
ENTITY_CAPACITY = 5
# Позиция двери в комнате.
DOOR_POS = {'left': (2, ROOM_SIZE[1] // 2), 'right': (ROOM_SIZE[0] - 3, ROOM_SIZE[1] //
2),
            'up': (ROOM_SIZE[0] // 2, 2), 'down': (ROOM_SIZE[0] // 2, ROOM_SIZE[1] - 3)}
# Положение портала относительно положения двери в комнате.
PORTAL_POS = {'left': (-1, 0), 'right': (1, 0), 'up': (0, -1), 'down': (0, 1)}
# Храним все изображения.
IMAGE = {}
# Храним все звуки.
SOUND = {}
TEST = {}
# Храним все состояние для существа.
LEGAL_STATE = ["stand_left", "stand_right", "move_up", "move_down", "move_right",
"move_left"]
# Хранилище анимации, в нем хранятся все анимации.
ANIMATION_REPOSITORY = None
# Инициализация изображения и звука, хранилища анимации.
class Init:
    global ANIMATION_REPOSITORY
    def __init__(self):
        self.scale = pygame.transform.scale
        self.load = pygame.image.load

    def init_creature(self, image, legal_state, name, scale_value):
        image[name] = {}
        for state in legal_state:
            image[name][state] = []
            i = 0
```

```
            while os.path.exists("data/image/creature/%s/%s/%d.png" % (name, state, i)):
                tmp_image = self.load("data/image/creature/%s/%s/%d.png" % (name, state,
i))
                tmp_width = round(scale_value * tmp_image.get_width())
                tmp_height = round(scale_value * tmp_image.get_height())
                image[name][state].append(self.scale(tmp_image, (tmp_width,
tmp_height)).convert_alpha())
                i += 1

    def init_music(sound, name, volume):
        pygame.mixer.init()
        pygame.mixer.music.load('data/music/%s.mp3' % name)
        pygame.mixer.music.set_volume(volume)
        pygame.mixer.music.play(-1)


    def init_obj(self, name, number, is_alpha=False):
        if is_alpha:
            IMAGE[name] = [self.load("data/image/%s/%d.png" % (name, i)).convert_alpha()
for i in range(number)]
        else:
            IMAGE[name] = [self.load("data/image/%s/%d.png" % (name, i)).convert() for i
in range(number)]

# Инициализация экрана
var.screen = pygame.display.set_mode(SCREEN_SIZE)

init = Init()
init.init_obj('ground', 8)
init.init_obj('wall', 6, is_alpha=True)
init.init_obj('chest', 6)
init.init_obj('potion', 3, is_alpha=True)
init.init_obj('trap', 6)
init.init_obj('door', 6, is_alpha=True)
init.init_obj('key', 1, is_alpha=True)
init.init_obj('bullet', 1, is_alpha=True)
init.init_obj('armor', 6, is_alpha=True)
init.init_obj('weapon', 5, is_alpha=True)
init.init_obj('treasure', 1, is_alpha=True)
init.init_obj('start', 13, is_alpha=True)
init.init_obj('pause', 6, is_alpha=True)
init.init_creature(IMAGE, LEGAL_STATE, "player", 2)
init.init_creature(IMAGE, LEGAL_STATE, "guard", 2)
init.init_creature(IMAGE, LEGAL_STATE, "mummy", 2)
init.init_creature(IMAGE, LEGAL_STATE, "pharaoh", 2)
Init.init_music(SOUND, "menu", 0.08)
ANIMATION_REPOSITORY = AnimationRepository()
```

var.py

```
"""
Хранит переменные, которые не являются постоянными.
Этот файл не должен включать какой-либо класс.
"""
# Список событий
key_up = []
key_down = []
mouse_up = []
mouse_down = []
# Вектор
mouse = None
# Поверхность
screen = None
# Интерфейс
interface = None
loading = None
start = None
play = None
death = None
end = None
pause = None
option = None
# Проверка на выход из игры
quit = False
# Карта
map = None
# Игрок
player = None
# Инвентарь
bag = None
# Число врагов по умолчанию
enemies_number = 3
```

interface_start.py

В main.py задается интерфейс начального меню var.interface = var.start = START() с изображением на заднем плане и тремя кнопками PLAY, SETTINGS, EXIT

```
from const import SCREEN_SIZE
from src.UI.interface import Interface
from src.UI.image_start import ImageStart
from src.UI.button_play import ButtonPlay
from src.UI.button_option import ButtonOption
from src.UI.button_quit import ButtonQuit

class START(Interface):
    def __init__(self):
        super().__init__(ImageStart((SCREEN_SIZE[0]/2, SCREEN_SIZE[1]/2)),
                         ButtonPlay((SCREEN_SIZE[0]/1.4, SCREEN_SIZE[1]/2 - 55)),
                         ButtonOption((SCREEN_SIZE[0]/1.4, SCREEN_SIZE[1]/2 + 30)),
                         ButtonQuit((SCREEN_SIZE[0]/1.4, SCREEN_SIZE[1]/2 + 115)))
```

При нажатии на кнопку SETTINGS мы заходим в интерфейс, показывающий нам текущие настройки управления, а также изменяемое количество врагов, которые будут появляться в комнатах в игре

button_option.py (пример одной из кнопок)

```
import var
import const
import pygame
from const import SCREEN_SIZE
from src.UI.button import Button
from src.UI.interface_option import OPTION


class ButtonOption(Button):
    def __init__(self, center):
        super().__init__(center, None,
pygame.transform.scale(const.IMAGE['start'][1], (SCREEN_SIZE[0] / 7.1, SCREEN_SIZE[1]
/ 20)), pygame.transform.scale(const.IMAGE['start'][5], (SCREEN_SIZE[0] / 7.1,
SCREEN_SIZE[1] / 20)))
# Смена фото кнопки с заливкой на без заливки при наведении
    def on_available(self):
        self.image = self.images[1]
    def on_hover(self):
        self.image = self.images[0]
# Так мы устанавливаем новый интерфейс при нажатии каждой кнопки
    def on_click(self):
        self.image = self.images[1]
        var.interface = var.option = OPTION()
```

button.py

```
import pygame
import pygame.display
import var
from pygame.surface import Surface
from src.UI.component import Component
from src.UI.label import Label
"""
Когда мышь не находится на ней, будет вызван on_available().
При клике по ней мышью будет вызван on_click().
Когда мышь находится на ней, будет вызван on_hover().
"""
class Button(Component):
    def __init__(self, center, label, image_active, image_inactive, mysize=None):
        self.label = label
        self.mysize = mysize
        super().__init__(center, image_active, image_inactive)

    @property
    def image(self):
        return self.__image

    @image.setter
    def image(self, value):
        if value != None and not isinstance(value, Surface):
            raise TypeError("Component.image must be Surface type.")
        center = self.rect.center
        if value == None:
            self.__image = None
            self.rect.size = self.mysize
        else:
            self.__image = value.copy()
            self.rect.size = self.__image.get_size()
        self.rect.center = center
        if self.label != None:
            self.label.draw(self.__image)

    @property
    def label(self):
        return self.__label

    @label.setter
    def label(self, value):
        if value != None and not isinstance(value, Label):
            raise TypeError("Button.label must be Label type.")
        self.__label = value

    def update(self):
        if self.rect.contain(var.mouse):
            if len(var.mouse_down) > 0:
                self.on_click()
            else:
                self.on_hover()
        else:
            self.on_available()

    def on_click(self):
        pass
    def on_hover(self):
        pass
    def on_available(self):
        pass
```

```
class CustomButton(Button):
    """
    Позволяет создать кнопку, просто используя цвет, без изображения.
    Имеет прямоугольную форму.
    """
    def __init__(self, center, button_active_color, button_inactive_color, button_size,
                 label_text, label_size, label_color=(0, 0, 0), label_font_type=None,
smooth=False):
        image_inactive = pygame.Surface(button_size).convert_alpha()
        image_active = pygame.Surface(button_size).convert_alpha()
        image_inactive.fill((0, 0, 0, 0))
        image_active.fill((0, 0, 0, 0))
        if not smooth:
            image_inactive.fill(button_active_color)
            image_active.fill(button_inactive_color)
        else:
            r = min(int(1/10*button_size[0]), int(1/10*button_size[1]))
            w = int(button_size[0])
            h = int(button_size[1])
            pygame.draw.circle(image_inactive, button_active_color, (r, r), r)
            pygame.draw.circle(image_inactive, button_active_color, (r, h - r), r)
            pygame.draw.circle(image_inactive, button_active_color, (w - r, r), r)
            pygame.draw.circle(image_inactive, button_active_color, (w - r, h - r), r)
            pygame.draw.rect(image_inactive, button_active_color, (0, r, w, h - 2 * r))
            pygame.draw.rect(image_inactive, button_active_color, (r, 0, w - 2 * r, h))
            pygame.draw.circle(image_active, button_inactive_color, (r, r), r)
            pygame.draw.circle(image_active, button_inactive_color, (r, h - r), r)
            pygame.draw.circle(image_active, button_inactive_color, (w - r, r), r)
            pygame.draw.circle(image_active, button_inactive_color, (w - r, h - r), r)
            pygame.draw.rect(image_active, button_inactive_color, (0, r, w, h - 2 * r))
            pygame.draw.rect(image_active, button_inactive_color, (r, 0, w - 2 * r, h))
        if label_text != '':
            label = Label((button_size[0]/2, button_size[1]/2), label_text, label_size,
label_color, label_font_type)
        else:
            label = None
        super().__init__(center, label, image_active, image_inactive)
```

component.py

```
from src.tool.rect import Rect
from pygame import Surface
"""
Базовый класс для отображения изображения кнопки.
"""
class Component:
    def __init__(self, center, *images):
        self.images = images
        self.rect = Rect()
        if len(self.images) == 0:
            self.image = None
        else:
            self.image = self.images[0]
            self.rect.size = self.image.get_size()
        self.rect.center = center

    @property
    def image(self):
        return self.__image

    @image.setter
    def image(self, value):
        if value != None and not isinstance(value, Surface):
            raise TypeError("Component.image must be Surface type.")
        center = self.rect.center
        if value == None:
            self.__image = None
            self.rect.size = (0, 0)
        else:
            self.__image = value.copy()
            self.rect.size = self.__image.get_size()
        self.rect.center = center

    @property
    def images(self):
        return self.__images

    @images.setter
    def images(self, value):
        self.__images = [_ for _ in value if isinstance(_, Surface)]

    @property
    def rect(self):
        return self.__rect

    @rect.setter
    def rect(self, value):
        self.__rect = Rect.init_one_arg(value)

    def draw(self, surface):
        if self.image != None:
            surface.blit(self.image, (int(self.rect.left), int(self.rect.top)))

    def update(self):
        pass
```

interface_loading.py

При нажатии на кнопку PLAY мы устанавливаем новый интерфейс: `var.interface = var.loading = LOADING()`, который начинает загрузку игры.

```
import var
from const import ROOM_SIZE, SCREEN_SIZE
from src.UI.progress_bar import ProgressBar
from src.UI.interface import Interface
from src.UI.label import Label
from src.bag import Bag
from src.map import Map
from src.obj.entity.creature.player import Player


class LOADING(Interface):
    def __init__(self):
        super().__init__(Label((SCREEN_SIZE[0]/2, SCREEN_SIZE[1]/2), ''),
ProgressBar((SCREEN_SIZE[0]/2, SCREEN_SIZE[1]/1.8), ((SCREEN_SIZE[0]/2.2,
SCREEN_SIZE[1]/54)), 5))
        self.__cnt = 0

    def update(self):
        if self.__cnt == 0:
            self.components[0].text = "Creating the player."
        elif self.__cnt == 1:
            self.components[0].text = "Creating the player backpack."
            var.player = Player((ROOM_SIZE[0] // 2, 3))
        elif self.__cnt == 2:
            self.components[0].text = "Loading the map."
            var.bag = Bag()
        elif self.__cnt == 3:
            self.components[0].text = "Put the player into the location."
            var.map = Map()
        elif self.__cnt == 4:
            self.components[0].text = "Done."
            var.map.active_room.entities.append(var.player)
        elif self.__cnt == 5:
            var.interface = var.play
        if self.__cnt == 5:
            self.__cnt = -1
        else:
            self.components[1].progress = self.__cnt
        self.__cnt += 1
```

Подгружаем состояние компонентов игры в var.py: Player, Bag, Map; добавляем сущность игрока на карту в комнату; изменяем интерфейс на новое состояние `var.interface = var.play`, которое уже инициализировано в main.py: `var.play = PLAY()`

interface_play.py

```
import pygame
import var
from const import SCREEN_SIZE
from src.UI.image_map import ImageMap
from src.UI.interface import Interface
from src.UI.label_bullet import LabelBullet
from src.UI.label_hp import LabelHP
from src.UI.label_hp_bar import LabelHPBar
from src.UI.label_bullet_bar import LabelBulletBar
from src.UI.progress_bar_bullet import ProgressBarBullet
from src.UI.progress_bar_hp import ProgressBarHP
from src.UI.label_debuff import LabelDebuffName, LabelDebuffTime
from src.UI.interface_pause import PAUSE
from src.UI.interface_death import DEATH
"""
Обновляем карту и выводим элементы интерфейса на экран.
Делаем проверки на состояние интерфейса (пауза) и игрока.
"""
class PLAY(Interface):
    def __init__(self):
        super().__init__(ImageMap((SCREEN_SIZE[0]/1.15, SCREEN_SIZE[1]/5)), LabelHP((20,
50)), ProgressBarHP((147, 48)), LabelHPBar((147, 50)),
                         LabelBullet((20, 80)), ProgressBarBullet((147, 78)),
LabelBulletBar((147, 80)),
                         LabelDebuffName((20, 110)), LabelDebuffTime((20, 140)))

    def update(self):
        var.map.update()
        super().update()
        for event in var.key_down:
            if event.key == pygame.K_ESCAPE:
                var.player.refresh()
                var.interface = var.pause = PAUSE()
                var.pause.refresh()
        if var.player.is_dead:
            var.interface = var.death = DEATH()

    def draw(self, surface):
        var.map.active_room.draw(surface)
        super().draw(surface)
```

bag.py

```
from src.obj.entity.item.armor import Armor
from src.obj.entity.item.armors import ArmorKevlar, ArmorPolice, ArmorTiger,
ArmorSapphire, ArmorSandstorm, ArmorAbsolute
from src.obj.entity.item.bullet import Bullet
from src.obj.entity.item.key import Key
from src.obj.entity.item.potion_hp import PotionHP
from src.obj.entity.item.potion_buff import PotionBuff
from src.obj.entity.item.weapon import Weapon
from src.obj.entity.item.weapons import Deagle, HKUMP, SawedOff, AK47, AWP
"""
Это наш инвентарь.
"""
class Bag:
    def __init__(self):
        # Выдаем игроку расходники
        self.__items = [Bullet(67), Key(3), PotionHP(3), PotionBuff(2)]
        self.__has_bullet = True
        self.__has_key = True
        self.__has_treasure = False
        self.active_item_id = 0
        # Выдаем игроку оружие и броню
        self.weapon = Deagle()
        self.armor = ArmorKevlar()

    @property
    def items(self):
        return self.__items

    @property
    def weapon(self):
        return self.__weapon

    @weapon.setter
    def weapon(self, value):
        if not isinstance(value, Weapon):
            raise TypeError("Bag.weapon must be weapon type.")
        self.__weapon = value

    @property
    def armor(self):
        return self.__armor

    @armor.setter
    def armor(self, value):
        if not isinstance(value, Armor):
            raise TypeError("Bag.armor must be Armor type.")
        self.__armor = value

    @property
    def active_item_id(self):
        return self.__active_item_id

    @active_item_id.setter
    def active_item_id(self, value):
        if not 0 <= value <= 7:
            raise ValueError("Bag.active_item_id must from 0 to 7.")
        self.__active_item_id = value

    @property
    def has_key(self):
        return self.__has_key
```

```
    @property
    def has_bullet(self):
        return self.__has_bullet

    @property
    def has_treasure(self):
        return self.__has_treasure

    def add_item(self, collectible):
        if isinstance(collectible, (Armor, Weapon)):
            if len(self.__items) < 8:
                self.__items.append(collectible)
                return True
            else:
                return False
        else:
            for item in self.__items:
                if item.name == collectible.name:
                    item.amount += collectible.amount
                    return True
            if len(self.__items) < 8:
                self.__items.append(collectible)
                if not self.has_key and collectible.name == 'Key':
                    self.__has_key = True
                if not self.has_bullet and collectible.name == 'Bullet':
                    self.__has_bullet = True
                if not self.has_treasure and collectible.name == 'Treasure':
                    self.__has_treasure = True
                return True
            else:
                return False

    def remove_item(self, collectible):
        if isinstance(collectible, (Armor, Weapon)):
            for _, item in enumerate(self.__items):
                if item.name == collectible.name:
                    self.__items.pop(_)
                    return True
        else:
            for _, item in enumerate(self.__items):
                if item.name == collectible.name:
                    if item.amount > collectible.amount:
                        item.amount -= collectible.amount
                        return True
                    elif item.amount == collectible.amount:
                        if item.name == 'Bullet':
                            self.__has_bullet = False
                        elif item.name == 'Key':
                            self.__has_key = False
                        elif item.name == 'Treasure':
                            self.__has_treasure = False
                        self.__items.pop(_)
                        return True
                    else:
                        return False
        return False
```

map.py

```
import random
import var
import pygame
from pygame.surface import Surface
from const import MAP_SIZE, DOOR_POS, PORTAL_POS, ROOM_SIZE
from src.obj.building.destination import Destination
from src.obj.building.door import Door
from src.obj.entity.item.treasure import Treasure
from src.room.room_empty import RoomEmpty
from src.room.room_path import RoomPath
from src.room.room_supply import RoomSupply
from src.room.room_maze import RoomMaze
from src.room.room_square import RoomSquare
from src.room.room_treasure import RoomTreasure
from src.room.room_h import RoomH
from src.room.room_t import RoomT
from src.room.room_x import RoomX
from src.room.room_trap import RoomTrap
from src.room.room_wall import RoomWall
"""
Map создает Room
Рандомно определяются расположения дверей, выбираются виды комнат.
Вводится обязательно начальная и конечная комната (с наградой).
"""
class Map:
    # Типы комнат, где первые две обязательные
    __ROOM_KIND = [RoomEmpty, RoomTreasure, RoomTrap, RoomT, RoomX,
                   RoomH, RoomSupply, RoomSquare, RoomMaze, RoomPath, RoomWall]

    def __init__(self):
        # Если число комнат меньше 20, то снова создается новая карта.
        while not self.__init_map():
            continue
        self.__active_room_pos = self.__entry_room_pos
        # The room which has been discover.
        self.__discover_set = set()
        self.__discover_set.add(self.__entry_room_pos)
        self.__surface = Surface((200, 200)).convert_alpha()
        self.__surface.fill((255, 255, 255, 100))

    @property
    def entry_room(self):
        return self.__map[self.__entry_room_pos[0]][self.__entry_room_pos[1]]

    @property
    def active_room(self):
        return self.__map[self.__active_room_pos[0]][self.__active_room_pos[1]]

    @property
    def end_room(self):
        return self.__map[self.__end_room_pos[0]][self.__end_room_pos[1]]


    def __init_map(self):
        self.__map = [[None for j in range(MAP_SIZE[1])] for i in range(MAP_SIZE[0])]
        self.__graph = [[[] for j in range(MAP_SIZE[1])] for i in range(MAP_SIZE[0])]
        self.__entry_room_pos = (random.randint(0, MAP_SIZE[0] - 1), 0)
        self.__end_room_pos = None
        end_rooms = []
        dire = [(0, -1), (0, 1), (-1, 0), (1, 0)]
        # dfs (Depth First Search - алгоритм поиска в глубину) для создания всей карты.
```

```
        def dfs(now, prev, graph, cnt_room, depth):
            if now[0] < 0 or now[0] >= MAP_SIZE[0] or \
                    now[1] < 0 or now[1] >= MAP_SIZE[1]:
                return 0
            if prev != now:
                graph[prev[0]][prev[1]].append(now)
                graph[now[0]][now[1]].append(prev)
            if random.randint(1, 2) == 1 and depth >= 9:
                end_rooms.append(now)
                return 1
            else:
                if depth >= 7 or len(end_rooms) <= 4 or cnt_room <= 25:
                    spread_path = random.randint(2, 3)
                else:
                    spread_path = 1
                random.shuffle(dire)
                for d in dire:
                    if spread_path == 0:
                        break
                    nxt = (now[0] + d[0], now[1] + d[1])
                    if nxt[0] < 0 or nxt[0] >= MAP_SIZE[0] or \
                            nxt[1] < 0 or nxt[1] >= MAP_SIZE[1]:
                        continue
                    if nxt in end_rooms or now in graph[nxt[0]][nxt[1]]:
                        continue
                    if len(graph[nxt[0]][nxt[1]]) == 0:
                        cnt_room += dfs(nxt, now, graph, cnt_room, depth + 1)
                        spread_path -= 1
                    else:
                        graph[nxt[0]][nxt[1]].append(now)
                        graph[now[0]][now[1]].append(nxt)
                    if len(graph[nxt[0]][nxt[1]]) == 1 and not (nxt in end_rooms):
                        end_rooms.append(nxt)
                return cnt_room
        if dfs(self.__entry_room_pos, self.__entry_room_pos, self.__graph, 0, 1) < 20:
            return False
        for i in range(MAP_SIZE[0]):
            for j in range(MAP_SIZE[1]):
                # Создаем комнату
                if len(self.__graph[i][j]) != 0:
                    # Начальная локация
                    if (i, j) == self.__entry_room_pos:
                        self.__map[i][j] = RoomEmpty()
                    # Конечная локация (с наградой)
                    elif (i, j) in end_rooms:
                        self.__map[i][j] = RoomTreasure()
                    # Другая комната
                    else:
                        self.__map[i][j] = Map.__ROOM_KIND[random.randint(2, 10)]()
                # Добавляем дверь/двери
                for item in self.__graph[i][j]:
                    if (item[0] - i, item[1] - j) == (0, -1):
                        to = 'up'
                    elif (item[0] - i, item[1] - j) == (0, 1):
                        to = 'down'
                    elif (item[0] - i, item[1] - j) == (-1, 0):
                        to = 'left'
                    elif (item[0] - i, item[1] - j) == (1, 0):
                        to = 'right'
                    else:
                        raise AttributeError("Illegal value of Map.__init_map.item in
graph[i][j].")
                    self.__map[i][j].add_door(to)
```

```
 # Выбираем конечная комнату и помещаем в нее сокровище.
        for end_room in end_rooms:
            if end_room[1] >= MAP_SIZE[1] // 2:
                self.__end_room_pos = end_room
        if self.__end_room_pos == None:
            end_room = end_rooms[random.randint(0, len(end_rooms) - 1)]
            self.__end_room_pos = end_room

        # Добавляем место для выхода из гробницы в начальную комнату
        self.entry_room.buildings[DOOR_POS['up'][0]][DOOR_POS['up'][1]] =
Destination(DOOR_POS['up'])
        return True


    def walk(self, to):
        """
        Изменяем отображения комнаты для игрока (портал)
        :param to: str. left, right, up, down.
        :return:
        """
        self.active_room.entities.remove(var.player)
        self.__active_room_pos = (self.__active_room_pos[0] + PORTAL_POS[to][0],
                                  self.__active_room_pos[1] + PORTAL_POS[to][1])
        self.__discover_set.add(self.__active_room_pos)
        for i in range(4):
            x, y = tuple(DOOR_POS.values())[i]
            dx, dy = tuple(PORTAL_POS.values())[i]
            if isinstance(self.active_room.buildings[x][y], Door) and \
               (self.__active_room_pos[0] + dx, self.__active_room_pos[1] + dy) in
self.__discover_set:
                self.active_room.buildings[x][y].can_access = True
                self.active_room.buildings[x][y].on_trigger(var.player)
        self.active_room.entities.append(var.player)
        # Если игрок вошел в нижнюю дверь, он появится у верхней двери и т.д.
        if to == 'left':
            to = 'right'
        elif to == 'right':
            to = 'left'
        elif to == 'up':
            to = 'down'
        elif to == 'down':
            to = 'up'
        var.player.rect.center =
self.active_room.buildings[DOOR_POS[to][0]][DOOR_POS[to][1]].rect.center


    def update(self):
        self.active_room.update()


    def to_surface(self, full_map=False):
        """
        Отображаем небольшой радар карты
        Помечаем закрытые и открытые двери, текущую комнату.
        :param full_map: Choose whether to display the whole map.
        :return: Surface
        """
        surface_size = 200, 200
        x, y = surface_size[0] / (MAP_SIZE[0] + 1), surface_size[1] / (MAP_SIZE[1] + 1)

        if full_map:
            pos_list = [(i, j) for i in range(MAP_SIZE[0]) for j in range(MAP_SIZE[1]) if
len(self.__graph[i][j]) != 0]
            starter = pos_list
```

```
        else:
            pos_list = self.__discover_set
        for pos in pos_list:
            left, top = (pos[0] + 2/3)*x, (pos[1] + 2/3)*y
            width, height = 2/3*x, 2/3*y
            pygame.draw.rect(self.__surface, (200, 200, 200), (round(left), round(top),
round(width), round(height)))
            dire = [(0, -1), (0, 1), (-1, 0), (1, 0)]
            for d in dire:
                nxt = pos[0] + d[0], pos[1] + d[1]
                if nxt in self.__graph[pos[0]][pos[1]]:
                    le, to = round(left + (1/6 + 1/2*d[0])*x), round(top + (1/6 +
1/2*d[1])*y)
                    wi, he = round(1/3*x), round(1/3*y)
                    if pos in self.__discover_set and nxt in self.__discover_set:
                        pygame.draw.rect(self.__surface, (40, 200, 40), (le, to, wi, he))
                    else:
                        pygame.draw.rect(self.__surface, (200, 40, 40), (le, to, wi, he))
        left, top = round((self.__active_room_pos[0] + 2/3) * x),
round((self.__active_room_pos[1] + 2/3) * y)
        width, height = round(2/3*x), round(2/3*y)
        pygame.draw.rect(self.__surface, (250, 210, 90), (left, top, width, height))
        return self.__surface
```

vector.py

```
import math
import random
"""
У вектора есть атрибуты x и y.
"""
class Vector:
    def __init__(self, x=0., y=0.):
        self.x = x
        self.y = y
        self.__stop = 0

    @classmethod
    def init_one_arg(cls, x_y):
        return cls(x_y[0], x_y[1])

    @classmethod
    def random_normalized_vector(cls):
        degree = random.randint(0, 359) / 180 * math.pi
        return cls(math.cos(degree), math.sin(degree))

    def copy(self):
        return Vector(self.x, self.y)

    @property
    def x(self):
        return self.__x

    @x.setter
    def x(self, value):
        self.__x = float(value)

    @property
    def y(self):
        return self.__y

    @y.setter
    def y(self, value):
        self.__y = float(value)

    @property
    def length(self):
        return math.sqrt(self.x * self.x + self.y * self.y)

    @length.setter
    def length(self, value):
        if value < 0:
            raise ValueError("The length of Vector can't less than zero.")
        length, value = self.length, float(value)
        if self.length == 0:
            return
        self.x *= value / length
        self.y *= value / length

    @property
    def sin(self):
        if self.length == 0:
            raise Exception("The sin for the Vector doesn't exist.")
        return self.y / self.length
```

```
    @property
    def cos(self):
        if self.length == 0:
            raise Exception("The cos for the Vector doesn't exist.")
        return self.x / self.length

    @property
    def tan(self):
        if self.x == 0:
            raise Exception("The sin for the Vector doesn't exist.")
        return self.y / self.x

    def __add__(self, vector):
        return Vector(self.x + vector[0], self.y + vector[1])

    def __sub__(self, vector):
        return Vector(self.x - vector[0], self.y - vector[1])

    def __mul__(self, number):
        number = float(number)
        return Vector(self.x*number, self.y*number)

    def __truediv__(self, number):
        number = float(number)
        return self*(1/number)

    def __iter__(self):
        return self

    def __next__(self):
        self.__stop += 1
        if self.__stop == 3:
            raise StopIteration
        return (None, self.x, self.y)[self.__stop]

    def __getitem__(self, item):
        if item == 0:
            return self.x
        elif item == 1:
            return self.y
        else:
            raise ValueError("Vector.__getitem__.item must be 0 or 1.")

    def rotate(self, angle):
        """
        Когда угол положительный, вектор вращается против часовой стрелки.
        :param angle: float or int.
        :return: Vector
        """
        if self.length == 0:
            return Vector(0, 0)
        angle = angle / 180 * math.pi
        x = self.y * math.cos(angle) + self.x * math.sin(angle)
        y = self.x * math.cos(angle) - self.y * math.sin(angle)
        return Vector(x, y)

    def dot(self, vector):
        return self.x * vector[0] + self.y * vector[1]

    def cross(self, vector):
        return self.x * vector[1] - self.y * vector[0]
```

```
def normalize(self):
    if self.length == 0:
        return self
    else:
        return self / self.length

def angle_to(self, vector):
    """
    Когда угол положительный, вектор вращается против часовой стрелки.
    :param vector: Vector
    :return: Vector
    """
    vector = Vector.init_one_arg(vector)
    if self.length == 0 or vector.length == 0:
        return 0
    sin = self.sin * vector.cos - self.cos * vector.sin
    cos = self.cos * vector.cos + self.sin * vector.sin
    if cos > 0:
        ans = math.asin(sin)
    elif sin >= 0:
        ans = math.acos(cos)
    else:
        ans = -math.acos(cos)
    ans = ans/math.pi*180
    return ans

def dis_to(self, vector):
    return (self - vector).length
```

rect.py

```
from src.tool.vector import Vector
"""
Это прямоугольник.
Его можно инициализировать четырьмя, двумя, одним аргументом.
Это можно повторить.
Он может проверить, находится ли в нем точка или она пересекается с другим
прямоугольником.
"""
class Rect:
    def __init__(self, left=0., top=0., width=0., height=0.):
        if width < 0:
            left += width
            width = -width
        if height < 0:
            top += height
            height = -height
        self.left = left
        self.top = top
        self.width = width
        self.height = height
        self.__stop = 0

    @classmethod
    def init_two_arg(cls, left_top, width_height):
        return cls(left_top[0], left_top[1], width_height[0], width_height[1])

    @classmethod
    def init_one_arg(cls, left_top_width_height):
        return cls(left_top_width_height[0], left_top_width_height[1],
left_top_width_height[2],
                   left_top_width_height[3])

    def copy(self):
        return Rect(self.left, self.top, self.width, self.height)

    def __iter__(self):
        return self

    def __next__(self):
        self.__stop += 1
        if self.__stop == 5:
            raise StopIteration
        return (None, self.left, self.top, self.width, self.height)[self.__stop]

    def __getitem__(self, item):
        return (self.left, self.top, self.width, self.height)[item]

    @property
    def left(self):
        return self.__left

    @left.setter
    def left(self, value):
        self.__left = float(value)

    @property
    def top(self):
        return self.__top
```

```
@top.setter
def top(self, value):
    self.__top = float(value)

@property
def width(self):
    return self.__width

@width.setter
def width(self, value):
    self.__width = float(value)

@property
def height(self):
    return self.__height

@height.setter
def height(self, value):
    self.__height = float(value)

@property
def right(self):
    return self.left + self.width

@right.setter
def right(self, value):
    self.left = float(value) - self.width

@property
def bottom(self):
    return self.top + self.height

@bottom.setter
def bottom(self, value):
    self.top = float(value) - self.height

@property
def center_x(self):
    return self.left + self.width / 2

@center_x.setter
def center_x(self, value):
    self.left = float(value) - self.width / 2

@property
def center_y(self):
    return self.top + self.height / 2

@center_y.setter
def center_y(self, value):
    self.top = float(value) - self.height / 2

@property
def size(self):
    return Vector(self.width, self.height)

@size.setter
def size(self, value):
    self.width = value[0]
    self.height = value[1]

@property
def left_top(self):
    return Vector(self.left, self.top)
```

```
    @left_top.setter
    def left_top(self, value):
        self.left = value[0]
        self.top = value[1]

    @property
    def left_bottom(self):
        return Vector(self.left, self.bottom)

    @left_bottom.setter
    def left_bottom(self, value):
        self.left = value[0]
        self.bottom = value[1]

    @property
    def right_top(self):
        return Vector(self.right, self.top)

    @right_top.setter
    def right_top(self, value):
        self.right = value[0]
        self.top = value[1]

    @property
    def right_bottom(self):
        return Vector(self.right, self.bottom)

    @right_bottom.setter
    def right_bottom(self, value):
        self.right = value[0]
        self.bottom = value[1]

    @property
    def center(self):
        return Vector(self.center_x, self.center_y)

    @center.setter
    def center(self, value):
        self.center_x = value[0]
        self.center_y = value[1]

    @property
    def left_center(self):
        return Vector(self.left, self.center_y)

    @left_center.setter
    def left_center(self, value):
        self.left = value[0]
        self.center_y = value[1]

    @property
    def right_center(self):
        return Vector(self.right, self.center_y)

    @right_center.setter
    def right_center(self, value):
        self.right = value[0]
        self.center_y = value[1]

    @property
    def top_center(self):
        return Vector(self.center_x, self.top)

    @top_center.setter
```

```
def top_center(self, value):
    self.center_x = value[0]
    self.top = value[1]

@property
def bottom_center(self):
    return Vector(self.center_x, self.bottom)

@bottom_center.setter
def bottom_center(self, value):
    self.center_x = value[0]
    self.bottom = value[1]

def contain(self, point):
    return self.left < point[0] < self.right and self.top < point[1] < self.bottom

def intersect(self, rect):
    if not isinstance(rect, Rect):
        raise TypeError('Rect.intersect.rect must be Rect type.')
    if self.left > rect.right or self.right < rect.left:
        return False
    elif self.top > rect.bottom or self.bottom < rect.top:
        return False
    else:
        return True
```

obj.py

```
from src.tool.rect import Rect
from src.tool.vector import Vector
from pygame import Surface
"""
Объект, который может отображаться на экране.
"""
class Obj:
    def __init__(self, rect, image, vector):
        """
        Используются заранее описанные инструменты vector, rect
        :param rect: Rect. Represent the position and the collide box for the obj.
        :param image: Image. The image the obj will display.
        :param vector: Vector. The relative position for the rect to draw the image.
        """
        self.rect = rect
        self.image = image
        self.vector = vector

    @property
    def rect(self):
        return self.__rect

    @rect.setter
    def rect(self, value):
        self.__rect = Rect.init_one_arg(value)

    @property
    def image(self):
        return self.__image

    @image.setter
    def image(self, value):
        if value != None and not isinstance(value, Surface):
            raise TypeError("Obj.image must be Surface or None type.")
        self.__image = value

    @property
    def vector(self):
        return self.__vector

    @vector.setter
    def vector(self, value):
        self.__vector = Vector(value[0], value[1])

    def draw(self, surface):
        if self.image != None:
            surface.blit(self.image, (int(self.rect.left + self.vector[0]),
int(self.rect.top + self.vector[1])))
```

entity.py

```
from const import TILE_SIZE
from src.obj.obj import Obj
from src.tool.rect import Rect
from src.tool.vector import Vector
"""
Сущность, которая может двигаться, сталкиваться со строениями.
update() вызывается каждый кадр.
"""
class Entity(Obj):
    def __init__(self, rect, image, vector, speed_dir, speed_mag):
        """
        :param rect: Rect.
        :param image: Image.
        :param vector: Vector.
        :param speed_dir: Vector. The direction the entity will move to.
        :param speed_mag: Float. The speed magnitude of the entity.
        """
        super().__init__(rect, image, vector)
        self.speed_dir = speed_dir
        self.speed_mag = speed_mag
        # Это кадр отображает изменение положения сущности.
        # Он может быть изменено только вызовом move().
        self.__delta_pos = Vector(0, 0)
        # В следующем кадре сущность переместится.
        self.__next_move = None

    @property
    def rect(self):
        return self.__rect

    @rect.setter
    def rect(self, value):
        value = Rect.init_one_arg(value)
        if value.width > TILE_SIZE[0] or value.height > TILE_SIZE[1]:
            raise ValueError("Creature.rect shouldn't greater than const.TILE_SIZE.")
        self.__rect = value

    @property
    def speed_dir(self):
        return self.__speed_dir

    @speed_dir.setter
    def speed_dir(self, value):
        self.__speed_dir = Vector.init_one_arg(value).normalize()

    @property
    def speed_mag(self):
        return self.__speed_mag

    @speed_mag.setter
    def speed_mag(self, value):
        if value < 0:
            raise ValueError("speed_mag can't less than zero.")
        self.__speed_mag = float(value)

    @property
    def delta_pos(self):
        return self.__delta_pos
```

```
    @property
    def next_move(self):
        return self.__next_move

    @next_move.setter
    def next_move(self, value):
        if value == None:
            self.__next_move = None
        else:
            self.__next_move = Vector.init_one_arg(value)

    def move(self, delta_pos=None):
        """
        Если delta_pos != None, он будет перемещаться на delta_pos.
        Иначе, если next_move != None, он переместится на next_move.
        Остальное перемещается в соответствии с speed_dir и speed_mag.
        :param delta_pos: Vector. The specified delta_pos
        :return:
        """
        if delta_pos != None:
            self.__delta_pos = Vector.init_one_arg(delta_pos)
        elif self.next_move != None:
            self.__delta_pos = self.next_move
            self.next_move = None
        else:
            self.__delta_pos = self.speed_dir * self.speed_mag
        self.rect.center += self.__delta_pos

    def collide_building(self, *buildings):
        """
        Обрабатываем столкновение со строениями.
        :param buildings: [Building]
        :return:
        """
        pass

    def collide_entity(self, *entities):
        """
        Обработайте столкновение с сущностями.
        :param entities: [Entity]
        :return:
        """
        pass

    def update(self):
        """
        Обрабатываем каждый кадр сущности.
        :return:
        """
        self.move()
```

item.py

```
from src.obj.entity.entity import Entity
from src.tool.vector import Vector
"""
Эта сущность не может контролироваться игроком или искусственным интеллектом.
Она передвигается автоматически.
У нее есть понятие ускорения.
"""
class Item(Entity):
    def __init__(self, rect, image, vector, speed_dir, speed_mag, accelerate_dir,
accelerate_mag):
        super().__init__(rect, image, vector, speed_dir, speed_mag)
        self.accelerate_dir = accelerate_dir
        self.accelerate_mag = accelerate_mag

    @property
    def accelerate_dir(self):
        return self.__accelerate_dir

    @accelerate_dir.setter
    def accelerate_dir(self, value):
        self.__accelerate_dir = Vector.init_one_arg(value).normalize()

    @property
    def accelerate_mag(self):
        return self.__accelerate_mag

    @accelerate_mag.setter
    def accelerate_mag(self, value):
        if value < 0:
            raise ValueError("accelerate_mag can't less than zero.")
        self.__accelerate_mag = float(value)

    def move(self, delta_pos=None):
        speed_vector = self.speed_dir * self.speed_mag
        accelerate_vector = self.accelerate_dir * self.accelerate_mag
        speed_vector = speed_vector + accelerate_vector
        self.speed_dir = speed_vector.normalize()
        self.speed_mag = speed_vector.length
        super().move(delta_pos)
```

collectible.py

```
import var
from src.obj.entity.item.item import Item
from src.tool.vector import Vector
from src.tool.rect import Rect
from src.obj.building.wall import Wall
"""
Когда игрок столкнется с предметом и нажмет клавишу пробела, он будет поднят в сумку, если
сумка не заполнена.
При выпадении из чего-либо совершает небольшое движение, может столкнуться со стеной.
В этом случае произойдет отскок и предмет не провалится в текстуру.
"""
class Collectible(Item):
    def __init__(self, name, image, can_use=False, amount=1):
        super().__init__(Rect(0, 0, 30, 30), image, Vector(-6, -6), Vector(0, 0), 8,
Vector(0, 0), 0)
        self.__name = name
        self.amount = amount
        self.friction_mag = 0.5
        self.can_use = can_use

    @property
    def name(self):
        return self.__name

    @property
    def amount(self):
        return self.__amount

    @amount.setter
    def amount(self, value):
        if value < 0:
            raise ValueError("Consumable.amount can't less than zero.")
        self.__amount = int(value)
        if self.__amount == 0:
            var.bag.remove_item(self)

    @property
    def friction_mag(self):
        return self.__friction_mag

    @friction_mag.setter
    def friction_mag(self, value):
        if value < 0:
            raise ValueError("friction_mag can't less than zero")
        self.__friction_mag = float(value)

    @property
    def information(self):
        """
        :return: str. This information will show on the bag when player select it.
        """
        return ""

    def move(self, delta_pos=None):
        if self.speed_mag == 0:
            return
        self.speed_mag = max(self.speed_mag - self.friction_mag, 0)
        super().move()
```

```
def explode(self, pos, speed_dir):
    self.rect.center = pos
    self.speed_dir = speed_dir
def collide_building(self, *buildings):
    if self.speed_dir.length == 0 or self.speed_mag == 0:
        return
    reflect_x, reflect_y = False, False
    for building in buildings:
        if not isinstance(building, Wall):
            continue
        if (self.speed_dir.x > 0 and self.rect.right >= building.rect.left) or \
                (self.speed_dir.x < 0 and self.rect.left <= building.rect.right):
            reflect_x = True
        if (self.speed_dir.y > 0 and self.rect.bottom >= building.rect.top) or \
                (self.speed_dir.y < 0 and self.rect.top <= building.rect.bottom):
            reflect_y = True
    if reflect_x:
        self.speed_dir.x *= -1
    if reflect_y:
        self.speed_dir.y *= -1

def use(self):
    self.amount -= 1
```

weapon.py

```
import var
from src.obj.entity.item.bullet import Bullet
from src.obj.entity.item.collectible import Collectible
from src.obj.entity.item.bubble_bullet import BubbleBullet
"""
Оно может стрелять и перезаряжаться.
Его можно использовать в виде снаряжения в инвентаре.
update() вызывается каждый кадр.
"""
class Weapon(Collectible):
    def __init__(self, name, image, damage, interval, clip, reload):
        super().__init__(name, image, True)
        self.damage = damage
        # Минимум кадров между выстрелами.
        self.interval = interval
        # Обойма с патронами.
        self.clip = clip
        # Время перезарядки, больше, чем интервал стрельбы.
        self.reload = reload
        # Счетчик для стрельбы, counter == -1 означает, что оружие готово к стрельбе.
        self.__counter = 0
        # Оставшиеся патроны в обойме, remain == 0 означает, что нужна перезарядка.
        self.remain = 0

    @property
    def damage(self):
        return self.__damage

    @damage.setter
    def damage(self, value):
        self.__damage = int(value)

    @property
    def interval(self):
        return self.__interval

    @interval.setter
    def interval(self, value):
        value = int(value)
        if value < 0:
            raise ValueError("Weapon.interval can't less than zero.")
        self.__interval = value

    @property
    def clip(self):
        return self.__clip

    @clip.setter
    def clip(self, value):
        value = int(value)
        if value == 0:
            raise ValueError("Weapon.clip can't be zero.")
        self.__clip = value
```

```
    @property
    def reload(self):
        return self.__reload

    @reload.setter
    def reload(self, value):
        value = int(value)
        if value < 0:
            raise ValueError("Weapon.reload can't less than zero.")
        self.__reload = value

    @property
    def remain(self):
        return self.__remain

    @remain.setter
    def remain(self, value):
        value = int(value)
        if value > self.clip:
            raise ValueError("Weapon.remain should be less than Weapon.clip.")
        self.__remain = value

    def update(self):
        super().update()
        # Перезарядка.
        if self.remain == 0 and self.__counter >= self.reload and var.bag.has_bullet:
            for item in var.bag.items:
                if item.name == 'Bullet':
                    self.remain = min(self.clip, item.amount)
                    item.amount -= self.remain
                    self.__counter = -1
        # Время подготовки к стрельбе.
        if self.remain != 0 and self.__counter >= self.interval:
            self.__counter = -1
        if self.__counter != -1:
            self.__counter += 1

    def reload_bullet(self):
        if 0 != self.remain != self.clip and self.__counter == -1 and var.bag.has_bullet:
            var.bag.add_item(Bullet(self.remain))
            self.__counter = 0
            self.remain = 0

    def shoot(self, pos, shoot_dir, shooter):
        if self.remain == 0 or self.__counter != -1:
            return None
        else:
            self.remain -= 1
            self.__counter = 0
            return BubbleBullet(pos, shoot_dir, 10, self.damage, shooter)

    def use(self):
        var.bag.remove_item(self)
        var.bag.add_item(var.bag.weapon)
        var.bag.weapon = self
```

weapons.py

```
import const
from src.obj.entity.item.weapon import Weapon


class Deagle(Weapon):
    def __init__(self):
        super().__init__("Desert Eagle", const.IMAGE['weapon'][0], 7, 50, 7, 90)

    @property
    def information(self):
        return '    Desert Eagle. pistol sometimes called a mini rifle, it has 7 points of
damage.'


class HKUMP(Weapon):
    def __init__(self):
        super().__init__("HK UMP", const.IMAGE['weapon'][1], 8, 8, 25, 80)

    @property
    def information(self):
        return '    Heckler & Koch UMP. fast but not so powerful SMG, it has 8 points of
damage.'


class SawedOff(Weapon):
    def __init__(self):
        super().__init__("Sawed Off", const.IMAGE['weapon'][2], 12, 70, 8, 130)

    @property
    def information(self):
        return '    Sawed Off. shotgun that can make anyone quiet in a second, it has 12
points of damage.'


class AK47(Weapon):
    def __init__(self):
        super().__init__("AK-47", const.IMAGE['weapon'][3], 9, 7, 30, 120)

    @property
    def information(self):
        return '    AK-47. soviet assault rifle, durable and glamorous, it has 9 points of
damage.'


class AWP(Weapon):
    def __init__(self):
        super().__init__("AWP", const.IMAGE['weapon'][4], 20, 100, 5, 160)

    @property
    def information(self):
        return '    Arctic Warfare Police. the best sniper rifle in the world, it has 20
points of damage.'
```

armor.py

```
import var
from src.obj.entity.item.collectible import Collectible
"""
У брони есть только один атрибут защиты.
Как и оружие, броню можно использовать в виде снаряжения в инвентаре.
"""
class Armor(Collectible):
    def __init__(self, name, image, defense):
        super().__init__(name, image, True)
        self.defense = defense

    @property
    def defense(self):
        return self.__defense

    @defense.setter
    def defense(self, value):
        if value <= 0:
            raise ValueError("Armor.defense can't less than zero.")
        self.__defense = int(value)

    def use(self):
        var.bag.remove_item(self)
        var.bag.add_item(var.bag.armor)
        var.bag.armor = self
```

armors.py

```
import const
from src.obj.entity.item.armor import Armor


class ArmorKevlar(Armor):
    def __init__(self):
        super().__init__('Kevlar vest', const.IMAGE['armor'][0], 3)

    @property
    def information(self):
        return '    Kevlar vest, it has 3 points of defence.'


class ArmorPolice(Armor):
    def __init__(self):
        super().__init__('Police black vest', const.IMAGE['armor'][1], 6)

    @property
    def information(self):
        return '    Police black vest, it has 6 points of defense.'


class ArmorTiger(Armor):
    def __init__(self):
        super().__init__('Tiger colored vest', const.IMAGE['armor'][2], 8)

    @property
    def information(self):
        return '    Tiger colored vest, it has 8 points of defense.'


class ArmorSapphire(Armor):
    def __init__(self):
        super().__init__('Sapphire vest', const.IMAGE['armor'][3], 10)

    @property
    def information(self):
        return '    Sapphire vest, it has 10 points of defense.'


class ArmorSandstorm(Armor):
    def __init__(self):
        super().__init__('Sandstorm vest', const.IMAGE['armor'][4], 12)

    @property
    def information(self):
        return '    Sandstorm vest, it has 12 points of defense.'


class ArmorAbsolute(Armor):
    def __init__(self):
        super().__init__('Absolute vest', const.IMAGE['armor'][5], 16)

    @property
    def information(self):
        return '    Absolute vest, it has 16 points of defense.'
```

creature.py

```
import var
from src.animation.animation_system import AnimationSystem
from src.debuff.debuff import Debuff
from src.obj.building.trigger_building import TriggerBuilding
from src.obj.entity.entity import Entity
from src.tool.vector import Vector
"""
У каждого существа есть здоровье, дебафф, урон, защита и система анимации.
Каждое существо может атаковать или получать урон.
"""
class Creature(Entity):
    def __init__(self, rect, animation_system, speed_mag, max_health, defense, damage):
        """
        :param rect: Rect
        :param animation_system: AnimationSystem
        :param speed_mag: float
        :param max_health: int
        :param defense: int
        :param damage: int
        """
        super().__init__(rect, None, None, Vector(0, 0), speed_mag)
        self.max_health = max_health
        self.defense = defense
        self.damage = damage
        self.shoot_dir = Vector(1, 0)
        self.health = self.max_health
        self.animation_system = animation_system
        self.animation_system.play("stand_left")
        # Счетчик получения урона.
        self.__cnt_take_damage = 0
        # Минимальное число кадров для нанесения урона.
        self.__interval_take_damage = 30
        # Debuff of the creature.
        self.__debuff = None

    @property
    def image(self):
        return self.__animation_system.image

    @image.setter
    def image(self, value):
        pass

    @property
    def vector(self):
        return self.__animation_system.vector

    @vector.setter
    def vector(self, value):
        pass

    @property
    def animation_system(self):
        return self.__animation_system

    @animation_system.setter
    def animation_system(self, value):
        if not isinstance(value, AnimationSystem):
            raise TypeError("Creature.animation_system must be AnimationSystem type.")
        self.__animation_system = value
```

```
@property
def max_health(self):
    return self.__max_health

@max_health.setter
def max_health(self, value):
    value = int(value)
    if value < 0:
        raise ValueError("Creature.max_health can't be less than zero.")
    self.__max_health = value

@property
def defense(self):
    return self.__defense

@defense.setter
def defense(self, value):
    value = int(value)
    if value < 0:
        raise ValueError("Creature.defense can't be less than zero.")
    self.__defense = value

@property
def damage(self):
    return self.__damage

@damage.setter
def damage(self, value):
    self.__damage = int(value)

@property
def shoot_dir(self):
    return self.__shoot_dir

@shoot_dir.setter
def shoot_dir(self, value):
    self.__shoot_dir = Vector.init_one_arg(value).normalize()

@property
def health(self):
    return self.__health

@health.setter
def health(self, value):
    value = int(value)
    self.__health = min(max(value, 0), self.max_health)

@property
def debuff(self):
    return self.__debuff

@debuff.setter
def debuff(self, value):
    if value != None and not isinstance(value, Debuff):
        raise TypeError("Creature.debuff must be Debuff or None type.")
    if value != None and self.debuff != None:
        return
    self.__debuff = value

@property
def is_dead(self):
    return self.health == 0
```

```
@property
def can_take_damage(self):
    return not self.is_dead and self.__cnt_take_damage >= self.__interval_take_damage

def reset_take_damage(self):
    """
    Вызываем при всех видах нанесения урона.
    :return:
    """
    self.__cnt_take_damage = 0

def update(self):
    # Обновляем состояние дебаффа (снимаем).
    if self.debuff != None:
        self.debuff.update()
        if self.debuff.time == 0:
            self.debuff.recover()
            self.debuff = None
    # Обновляем счетчик нанесения урона.
    self.__cnt_take_damage += 1
    # Обновляем состояния движения и атаки
    self.move()
    self.attack()
    # Выбор, какая анимация будет воспроизводиться с помощью speed_dir и shoot_dir.
    if self.speed_dir.length == 0:
        if 0 <= self.shoot_dir.angle_to(Vector(0, 1)) < 180:
            self.__animation_system.play("stand_left")
        else:
            self.__animation_system.play("stand_right")
    elif 0 <= self.speed_dir.angle_to(Vector(0, 1)) < 180:
        self.__animation_system.play("move_left")
    else:
        self.__animation_system.play("move_right")
    # Обновляем систему анимации.
    self.__animation_system.update()
    # Проверка на смерть.
    if self.is_dead:
        var.map.active_room.entities.remove(self)

def attack(self):
    pass

def take_bullet_damage(self, shootingbullet):
    if not self.can_take_damage:
        return
    damage = shootingbullet.damage
    if damage > 0:
        damage = max(damage - self.defense, 1)
    self.health -= damage
    # Урон от пули заставит существо переместиться в направлении пули.
    self.next_move = shootingbullet.speed_dir * 10
    self.reset_take_damage()
    try:
        var.map.active_room.entities.remove(shootingbullet)
    except ValueError:
        pass

def take_creature_damage(self, creature):
    pass
```

```
    def take_damage(self, damage):
        """
        Непосредственно получаем урон.
        :param damage: int
        :return:
        """
        if self.can_take_damage:
            self.health -= damage
            self.reset_take_damage()

    def collide_building(self, *buildings):
        # Флаг, который проверяет, может ли объект переместиться.
        can_access = True

        for building in buildings:
            # Если объект не двигается, идем дальше.
            if self.delta_pos.length == 0:
                break
            # Строение недостижимо, и объект сталкивается с ним.
            if not building.can_access and self.rect.intersect(building.rect):
                can_access = False
                # Сначала отменяем перемещение.
                delta_pos = self.delta_pos
                self.move(delta_pos * -1)

                # Пробуем двигаться только в направлении x или y, смотрим, не сталкиваемся
ли все еще со строением.
                # Если это так, просто установим это направление перемещения равным нулю.
                x, y = 1, 1
                # Пробуем двигаться только в направлении y.
                self.move((0, delta_pos[1]))
                if self.rect.intersect(building.rect):
                    y = 0
                self.move((0, delta_pos[1] * -1))
                # Пробуем двигаться только в направлении x.
                self.move((delta_pos[0], 0))
                if self.rect.intersect(building.rect):
                    x = 0
                self.move((delta_pos[0] * -1, 0))
                # Теперь двигаемся в соответствии с x и y.
                self.move((delta_pos[0] * x, delta_pos[1] * y))

            if isinstance(building, TriggerBuilding):
                building.on_trigger(self)
        # Обрабатываем особый случай столкновения.
        # Два прямоугольника имеют перекрывающиеся углы.
        # В этом случае сущность может перемещаться в направлении x или y, не сталкиваясь
с прямой.
        # Но если они будут двигаться вместе, они столкнутся, отменяем это движение.
        if not can_access and self.delta_pos[0] != 0 and self.delta_pos[1] != 0:
            self.move(self.delta_pos * -1)
            self.move((0, 0))
```

monster.py

```
import random
import var
from src.tool.vector import Vector
from src.obj.entity.creature.creature import Creature
from src.obj.entity.item.collectible import Collectible
from src.obj.entity.item.shooting_bullet import ShootingBullet
from src.obj.entity.item.weapons import Deagle, HKUMP, SawedOff, AK47
from src.obj.entity.item.armors import ArmorPolice, ArmorTiger, ArmorSapphire,
ArmorSandstorm
from src.obj.entity.item.bullet import Bullet
from src.obj.entity.item.potion_hp import PotionHP
from src.obj.entity.item.potion_buff import PotionBuff
from src.obj.entity.item.treasure import Treasure
"""
Монстр будет атаковать игрока.
При убийстве монстра выпадает лут (Collectible instances).
"""
class Monster(Creature):
    def __init__(self, rect, animation_system, speed_mag, max_health, defense, damage,
*collectibles):
        super().__init__(rect, animation_system, speed_mag, max_health, defense, damage)
        self.__collectibles = [*collectibles]
        # Добавляем возможные расходники, оружие и броню «в монстров»
        self.__collectibles.append(Bullet(random.randint(2, 3) * 8))
        if random.randint(0, 1) == 0:
            self.__collectibles.append(PotionHP())
        if random.randint(0, 4) == 0:
            self.__collectibles.append(PotionBuff())

        if random.randint(0, 4) == 0:
            self.__collectibles.append(Deagle())
        elif random.randint(0, 4) == 0:
            self.__collectibles.append(HKUMP())
        elif random.randint(0, 6) == 0:
            self.__collectibles.append(SawedOff())
        elif random.randint(0, 8) == 0:
            self.__collectibles.append(AK47())

        if random.randint(0, 4) == 0:
            self.__collectibles.append(ArmorPolice())
        elif random.randint(0, 4) == 0:
            self.__collectibles.append(ArmorTiger())
        elif random.randint(0, 6) == 0:
            self.__collectibles.append(ArmorSapphire())
        elif random.randint(0, 8) == 0:
            self.__collectibles.append(ArmorSandstorm())
        # Add the treasure
        elif random.randint(0, 50) == 0:
            self.__collectibles.append(Treasure())

    @property
    def collectibles(self):
        return self.__collectibles

    @collectibles.setter
    def collectibles(self, value):
        for collectible in value:
            if not isinstance(collectible, Collectible):
                raise TypeError("Monster.collectibles must be a list of Collectible type
instance.")
        self.__collectibles = value
```

```
def update(self):
    super().update()
    # Если монстр убит, с него выпадают предметы.
    if self.is_dead:
        for collectible in self.__collectibles:
            collectible.explode(self.rect.center, Vector.random_normalized_vector())
            var.map.active_room.entities.append(collectible)

@property
def can_see_player(self):
    return True

def collide_entity(self, *entities):
    for entity in entities:
        if isinstance(entity, ShootingBullet) and entity.owner == var.player:
            self.take_bullet_damage(entity)
        elif entity == var.player:
            var.player.collide_entity(self)
```

mummy.py (пример врага, ближний бой)

```
import const
import var
from const import TILE_SIZE
from src.animation.animation_system import AnimationSystem
from src.obj.entity.creature.monster import Monster
from src.tool.rect import Rect
from src.tool.vector import Vector
"""
Мумия будет вечно подходить к игроку с небольшой скоростью и бить, пока не умрет.
"""
class Mummy(Monster):
    def __init__(self, pos, *collectibles):
        super().__init__(Rect(pos[0] * TILE_SIZE[0], pos[1] * TILE_SIZE[1], 35, 20),
                         AnimationSystem(**const.ANIMATION_REPOSITORY.animations['mummy'])
, 2, 13, 5, 4, *collectibles)
        self.rect.center = self.rect.left_top + Vector(TILE_SIZE[0] / 2, TILE_SIZE[1] / 2)
        self.move_interval = 100
        self.move_time = 0
        self.Vector = Vector(0, 0)

    @property
    def speed_dir(self):
        if self.move_time == self.move_interval:
            self.move_time = 0
            self.Vector = (var.player.rect.center - self.rect.center).normalize()
        else:
            self.move_time += 1
        return self.Vector

    @speed_dir.setter
    def speed_dir(self, value):
        value = Vector.init_one_arg(value)
        if value.length != 0:
            raise AttributeError("Pharaoh.speed_dir can't be set.")
```

pharaoh.py (пример врага, дальний бой)

```
import const
import random
import var
from const import TILE_SIZE
from src.animation.animation_system import AnimationSystem
from src.obj.entity.creature.monster import Monster
from src.tool.rect import Rect
from src.tool.vector import Vector
from src.obj.entity.item.bubble_bullet import BubbleBullet
"""
Фараон будет стрелять в игрока и рандомно перемещаться, что делает его сложным для
попадания врагом, но в то же время с небольшим числом очков здоровья.
"""
class Pharaoh(Monster):
    def __init__(self, pos, *collectibles):
        super().__init__(Rect(pos[0] * TILE_SIZE[0], pos[1] * TILE_SIZE[1], 28, 18),
                         AnimationSystem(**const.ANIMATION_REPOSITORY.animations['pharaoh'
]), 2, 9, 3, 2, *collectibles)
        self.rect.center = self.rect.left_top + Vector(TILE_SIZE[0] / 2, TILE_SIZE[1] / 2)
        self.move_interval = 200
        self.move_time = 0
        self.Vector = Vector(0, 0)
        self.shoot_interval = 80
        self.shoot_time = 0
        self.shoot_damage = 2

    @property
    def shoot_dir(self):
        return (var.player.rect.center - self.rect.center).normalize()

    @shoot_dir.setter
    def shoot_dir(self, value):
        pass

    @property
    def speed_dir(self):
        if self.move_time == self.move_interval:
            vector = Vector(0, 0)
            vector.x += random.randint(-1, 1)
            vector.y += random.randint(-1, 1)
            self.move_time = 0
            self.Vector = vector.normalize()
        else:
            self.move_time += 1
        return self.Vector

    @speed_dir.setter
    def speed_dir(self, value):
        value = Vector.init_one_arg(value)
        if value.length != 0:
            raise AttributeError("Pharaoh.speed_dir can't be set.")

    def attack(self):
        if self.shoot_time > self.shoot_interval:
            shooting_bullet = BubbleBullet(self.rect.center, self.shoot_dir, 5,
self.shoot_damage, self)
            self.shoot_time = 0
            var.map.active_room.entities.append(shooting_bullet)
        else:
            self.shoot_time += 1
```

player.py

```
import var
import pygame
import const
from const import TILE_SIZE
from src.animation.animation_system import AnimationSystem
from src.obj.entity.creature.creature import Creature
from src.obj.entity.creature.monster import Monster
from src.obj.entity.item.collectible import Collectible
from src.obj.entity.item.shooting_bullet import ShootingBullet
from src.tool.rect import Rect
from src.tool.vector import Vector
"""
Это игрок.
Контроль осуществляется мышкой и клавиатурой.
Здесь же описаны различные характеристики оружия (а также звуки) и брони.
"""

class Player(Creature):
    def __init__(self, pos):
        super().__init__(Rect(pos[0] * TILE_SIZE[0], pos[1] * TILE_SIZE[1], 24, 10),
                         AnimationSystem(**const.ANIMATION_REPOSITORY.animations['player']
), 4, 10, 0, 0)
        self.rect.center = self.rect.left_top + Vector(TILE_SIZE[0] / 2, TILE_SIZE[1] / 2)
        # Нажимается соответствующая клавиша на клавиатуре.
        self.__w, self.__s, self.__a, self.__d,  self.__r, self.__v, self.__space = False,
False, False, False, False, False, False, False
        # Нажимается левая кнопка мыши.
        self.__b1 = False

        self.shoot_deagle = pygame.mixer.Sound('data/music/deagle.mp3')
        self.shoot_ump = pygame.mixer.Sound('data/music/ump.mp3')
        self.shoot_sawedoff = pygame.mixer.Sound('data/music/sawedoff.mp3')
        self.shoot_ak47 = pygame.mixer.Sound('data/music/ak47.mp3')
        self.shoot_awp = pygame.mixer.Sound('data/music/awp.mp3')


    @property
    def defense(self):
        return var.bag.armor.defense

    @defense.setter
    def defense(self, value):
        pass

    @property
    def damage(self):
        return var.bag.weapon.damage

    @damage.setter
    def damage(self, value):
        pass

    @property
    def shoot_dir(self):
        return (var.mouse - Vector.init_one_arg(var.map.active_room.blit_point) -
self.rect.center).normalize()

    @shoot_dir.setter
    def shoot_dir(self, value):
        pass
```

```
def update(self):
    # Обновляем клавиши.
    for event in var.key_down:
        if event.key == pygame.K_w or event.key == pygame.K_UP:
            self.__w = True
        elif event.key == pygame.K_s or event.key == pygame.K_DOWN:
            self.__s = True
        elif event.key == pygame.K_a or event.key == pygame.K_LEFT:
            self.__a = True
        elif event.key == pygame.K_d or event.key == pygame.K_RIGHT:
            self.__d = True
        elif event.key == pygame.K_r:
            self.__r = True
        elif event.key == pygame.K_v:
            self.__v = True
        elif event.key == pygame.K_SPACE:
            self.__space = True
    for event in var.key_up:
        if event.key == pygame.K_w or event.key == pygame.K_UP:
            self.__w = False
        elif event.key == pygame.K_s or event.key == pygame.K_DOWN:
            self.__s = False
        elif event.key == pygame.K_a or event.key == pygame.K_LEFT:
            self.__a = False
        elif event.key == pygame.K_d or event.key == pygame.K_RIGHT:
            self.__d = False
        elif event.key == pygame.K_v:
            self.__v = False
        elif event.key == pygame.K_SPACE:
            self.__space = False
    for event in var.mouse_down:
        if event.button == 1:
            self.__b1 = True
    for event in var.mouse_up:
        if event.button == 1:
            self.__b1 = False
    # Обновляем положение игрока
    super().update()
    # Перезарядка
    if self.__r:
        var.bag.weapon.reload_bullet()
        self.__r = False
    # Обновляем оружие
    var.bag.weapon.update()

@property
def speed_dir(self):
    vector = Vector(0, 0)
    if self.__w:
        vector.y -= 1
    if self.__s:
        vector.y += 1
    if self.__a:
        vector.x -= 1
    if self.__d:
        vector.x += 1
    return vector.normalize()

@speed_dir.setter
def speed_dir(self, value):
    value = Vector.init_one_arg(value)
    if value.length != 0:
        raise AttributeError("Player.speed_dir can't be set.")
```

```
    def attack(self):
        if self.__space or self.__b1:
            shooting_bullet = var.bag.weapon.shoot(self.rect.center, self.shoot_dir, self)
            if shooting_bullet != None:
                if var.bag.weapon.name == "Desert Eagle":
                    self.shoot_deagle.play().set_volume(0.35)
                elif var.bag.weapon.name == "HK UMP":
                    self.shoot_ump.play().set_volume(0.2)
                elif var.bag.weapon.name == "Sawed Off":
                    self.shoot_sawedoff.play().set_volume(0.04)
                elif var.bag.weapon.name == "AK-47":
                    self.shoot_ak47.play().set_volume(0.08)
                elif var.bag.weapon.name == "AWP":
                    self.shoot_awp.play().set_volume(0.1)
                var.map.active_room.entities.append(shooting_bullet)


    def collide_entity(self, *entities):
        for entity in entities:
            # Подбираем Collectibles
            if isinstance(entity, Collectible) and self.__v and var.bag.add_item(entity):
                var.map.active_room.entities.remove(entity)
            # Получаем урон от пули
            elif isinstance(entity, ShootingBullet) and entity.owner != self:
                self.take_bullet_damage(entity)
            # Получаем урон от ближнего боя
            elif isinstance(entity, Monster):
                self.take_creature_damage(entity)

    def refresh(self):
        # Обновляем нажатие клавиш, так как иногда при нажатии в интерфейсе PLAY, нажатие
может сработать в других интерфейсах.
        self.__w, self.__s, self.__a, self.__d,  self.__v, self.__space, self.__b1 =
(False for _ in range(7))

    def take_creature_damage(self, creature):
        if not self.can_take_damage:
            return
        damage = creature.damage
        if damage > 0:
            damage = max(damage - self.defense, 1)
        self.health -= damage
        self.reset_take_damage()
```

building.py

```
import const
from src.obj.obj import Obj
from src.tool.rect import Rect
from src.tool.vector import Vector
"""
Строение - это объект, который не изменяет своего положения.
Это поле столкновения зависит от const.TILE_SIZE.
Атрибут can_destroy пока не имеет смысла, возможно в будущем будет реализовано разрушение.
"""
class Building(Obj):
    def __init__(self, pos, image=None, vector=(0, 0), can_access=False,
can_destroy=False):
        """
        :param pos: Vector. Represent the position on the room, in Tile level not on pixel
level.
        :param image: Surface
        :param vector: Vector.
        :param can_access: bool
        :param can_destroy: bool
        """
        super().__init__(Rect(0, 0, const.TILE_SIZE[0], const.TILE_SIZE[1]), image,
vector)
        self.pos = pos
        self.can_access = can_access
        self.can_destroy = can_destroy

    @property
    def pos(self):
        return self.__pos

    @pos.setter
    def pos(self, value):
        self.__pos = Vector.init_one_arg(value)
        self.rect.left_top = (self.__pos[0] * const.TILE_SIZE[0], self.__pos[1] *
const.TILE_SIZE[1])

    @property
    def can_access(self):
        return self.__can_access

    @can_access.setter
    def can_access(self, value):
        if not isinstance(value, bool):
            raise TypeError("Building.can_access must be bool type.")
        self.__can_access = value

    @property
    def can_destroy(self):
        return self.__can_destroy

    @can_destroy.setter
    def can_destroy(self, value):
        if not isinstance(value, bool):
            raise TypeError("Building.can_destroy must be bool type.")
        self.__can_destroy = value
```

trigger_building.py

```
from src.obj.building.building import Building
"""
Когда игрок столкнется с ним, будет вызван on_trigger().
new_image, new_vector, old_image, old_vector просто используются для хранения изображения
и delta_pos.
На текущем этапе игры они не нужны, можно просто указать None для new_image и (0, 0) для
new_vector.
"""
class TriggerBuilding(Building):
    def __init__(self, pos, image, vector, can_access, can_destroy,
                 new_image, new_vector):
        super().__init__(pos, new_image, new_vector, can_access, can_destroy)
        self.can_trigger = True
        self.__new_image = self.image
        self.__new_vector = self.vector
        self.__old_image = self.image = image
        self.__old_vector = self.vector = vector

    @property
    def can_trigger(self):
        return self.__can_trigger

    @can_trigger.setter
    def can_trigger(self, value):
        if not isinstance(value, bool):
            raise TypeError("TriggerBuilding.can_trigger must be bool type.")
        self.__can_trigger = value

    @property
    def new_image(self):
        return self.__new_image

    @property
    def new_vector(self):
        return self.__new_vector

    @property
    def old_image(self):
        return self.__old_image

    @property
    def old_vector(self):
        return self.__old_vector

    def update(self):
        pass

    def on_trigger(self, entity):
        pass
```

bubble_sort.py

```
"""
Пузырьковая сортировка.
При обновлении сущностей их относительный порядок не будет слишком сильно меняться между
каждым кадром.
Таким образом, bubble_sort для сущностей - хороший выбор.
Работает почти без заддержки.
"""
def bubble_sort(_list, _compare):
    for i in range(len(_list)):
        for j in reversed(range(1, i + 1)):
            if _compare(_list[j], _list[j - 1]) < 0:
                _list[j], _list[j - 1] = _list[j - 1], _list[j]
            else:
                break
```

quadtree.py

```
from const import ENTITY_CAPACITY
from src.tool.rect import Rect
"""
Дерево квадрантов. Используется для уменьшения времени проверки на столкновение.
"""
class QuadTree:
    def __init__(self, rect):
        self.__rect = rect
        self.__entities = []
        self.__children = None

    @property
    def rect(self):
        return self.__rect

    @property
    def left_top_rect(self):
        return Rect.init_two_arg(self.rect.left_top, self.rect.size/2)

    @property
    def right_top_rect(self):
        return Rect.init_two_arg(self.rect.top_center, self.rect.size/2)

    @property
    def left_bottom_rect(self):
        return Rect.init_two_arg(self.rect.left_center, self.rect.size/2)

    @property
    def right_bottom_rect(self):
        return Rect.init_two_arg(self.rect.center, self.rect.size/2)

 def __collide_child(self, entity):
        """
        Смотрим, может ли сущность полностью войти в какой-либо из дочерних элементов.
        :param entity: Entity.
        :return: int or None. The index in the __children list. None represent it can't
put into any children rect.
        """
        if entity.rect.right < self.rect.center_x:
            if entity.rect.bottom < self.rect.center_y:
                return 0
            elif entity.rect.top > self.rect.center_y:
                return 2
            else:
                return None
        elif entity.rect.left > self.rect.center_x:
            if entity.rect.bottom < self.rect.center_y:
                return 1
            elif entity.rect.top > self.rect.center_y:
                return 3
            else:
                return None
        else:
            return None

 def add(self, entity):
        """
        Add an entity into the Tree.
        :param entity: Entity
        :return:
        """
```

```
        collide_child = self.__collide_child(entity)
        # Если сущность не может вместить ни в один из дочерних элементов.
        # Тогда помещаем в новую запись.
        if collide_child == None:
            self.__entities.append(entity)
            return
        # Если children нет.
        if self.__children == None:
            # Если количество сущностей больше или равно вместимости одной записи.
            if len(self.__entities) >= ENTITY_CAPACITY:
                self.__children = [QuadTree(self.left_top_rect),
QuadTree(self.right_top_rect),
                                   QuadTree(self.left_bottom_rect),
QuadTree(self.right_bottom_rect)]
                # Помещаем в child.
                self.__children[collide_child].add(entity)
                # Помещаем исходные сущности в записи в дочерний элемент, если возможно.
                for entity in self.__entities:
                    collide_child = self.__collide_child(entity)
                    if collide_child != None:
                        self.__children[collide_child].add(entity)
                        self.__entities.remove(entity)
            # Помещаем в эту запись.
            else:
                self.__entities.append(entity)
        # Помещаем в child.
        else:
            self.__children[collide_child].add(entity)

    def qry(self, entity):
        """
        Запрашиваем сущность, которая может столкнуться с имеющейся сущностью.
        :param entity: Entity
        :return: [Entity]
        """
        entities = []
        if self.__children != None:
            collide_child = self.__collide_child(entity)
            # Если столкновение не только с одним дочерним элементом.
            if collide_child == None:
                # Проверяем дочерний элементом и добавляем сущности.
                if entity.rect.left < self.rect.center_x:
                    if entity.rect.top < self.rect.center_y:
                        entities += self.__children[0].__get_all_entities()
                    if entity.rect.bottom > self.rect.center_y:
                        entities += self.__children[2].__get_all_entities()
                if entity.rect.right > self.rect.center_x:
                    if entity.rect.top < self.rect.center_y:
                        entities += self.__children[1].__get_all_entities()
                    if entity.rect.bottom > self.rect.center_y:
                        entities += self.__children[3].__get_all_entities()
            else:
                entities += self.__children[collide_child].qry(entity)
        # Добавляем сущностей в запись.
        entities += self.__entities
        return entities

    def __get_all_entities(self):
        entities = []
        if self.__children != None:
            for child in self.__children:
                entities += child.__get_all_entities()
        entities += self.__entities
        return entities
```

room.py

```
import random
from pygame.surface import Surface
import const
import var
from const import ROOM_SIZE, TILE_SIZE, IMAGE, DOOR_POS, PORTAL_POS
from src.obj.building.building import Building
from src.obj.building.door import Door
from src.obj.building.ground import Ground
from src.obj.building.portal import Portal
from src.obj.building.trigger_building import TriggerBuilding
from src.obj.building.wall import Wall
from src.obj.entity.creature.creature import Creature
from src.obj.entity.creature.guard import Guard
from src.obj.entity.creature.monster import Monster
from src.obj.entity.creature.mummy import Mummy
from src.obj.entity.creature.pharaoh import Pharaoh
from src.obj.entity.item.collectible import Collectible
from src.obj.entity.item.key import Key
from src.tool.bubble_sort import bubble_sort
from src.tool.quadtree import QuadTree
from src.tool.rect import Rect
"""
init_buildings() и init_entities() определяют, как выглядит комната и какие монстры в ней
есть.
Этот класс обновляет все, что отображается на экране.
Обновление триггера строений, сущности.
Проверка столкновения между объектом и зданием.
Проверка на столкновение между сущностью и сущностью. Используя QuadTree.
"""
class Room:
    def __init__(self):
        # Храним строения. __buildings[i][j] представляет определенное здание.
        self.__buildings = [[Building((i, j)) for j in range(ROOM_SIZE[1])] for i in
range(ROOM_SIZE[0])]
        # Храним объекты в Room.
        self.__entities = []
        # Храним триггеры строений в Room.
        self.__trigger_buildings = []
        # Инициализия Room.
        self.init_buildings()
        self.init_entities()
        # Добавляем триггеры строений.
        for _ in self.__buildings:
            for building in _:
                if isinstance(building, TriggerBuilding):
                    self.__trigger_buildings.append(building)
        # Room отображает все вещи на __surface и возвращает их в интерфейс, показывая на
экране.
        self.__surface = Surface((TILE_SIZE[0] * ROOM_SIZE[0], TILE_SIZE[1] *
ROOM_SIZE[1]))
        self.__surface.set_colorkey((0, 0, 0))
        # Точка на экране используется для расчета направления выстрела игрока.
        self.__blit_point = (0, 0)

    @property
    def buildings(self):
        return self.__buildings

    @property
    def entities(self):
        return self.__entities
```

```
    @property
    def blit_point(self):
        return self.__blit_point

    def init_buildings(self):
        """
        Инициализация стены вокруг комнаты и почвы внутри комнаты.
        :return:
        """
        for i, row in enumerate(self.buildings):
            for j, col in enumerate(row):
                if i == 2 and 2 <= j <= ROOM_SIZE[1] - 4:
                    self.buildings[i][j] = Wall((i, j), IMAGE['wall'][0], (0, -12))
                elif i == 2 and j == ROOM_SIZE[1] - 3:
                    self.buildings[i][j] = Wall((i, j), IMAGE['wall'][4], (0, -12))
                elif i == ROOM_SIZE[0] - 3 and 2 <= j <= ROOM_SIZE[1] - 4:
                    self.buildings[i][j] = Wall((i, j), IMAGE['wall'][1], (0, -12))
                elif i == ROOM_SIZE[0] - 3 and j == ROOM_SIZE[1] - 3:
                    self.buildings[i][j] = Wall((i, j), IMAGE['wall'][3], (0, -12))
                elif (j == 2 or j == ROOM_SIZE[1] - 3) and 2 <= i <= ROOM_SIZE[0] - 3:
                    self.buildings[i][j] = Wall((i, j), IMAGE['wall'][2], (0, -12))
                elif 2 < i < ROOM_SIZE[0] - 3 and 2 < j < ROOM_SIZE[1] - 3:
                    self.buildings[i][j] = Ground((i, j),
IMAGE['ground'][random.randint(0, 7)])


    def init_entities(self):
        """
        Случайным образом добавляем монстра в комнату на земле.
        У одного монстра среди них будет ключ.
        :return:
        """
        cnt = 0
        while cnt != var.enemies_number:
            pos = (random.randint(0, ROOM_SIZE[0] - 1), random.randint(0, ROOM_SIZE[1] -
1))
            if not isinstance(self.buildings[pos[0]][pos[1]], Ground):
                continue
            have_key = random.randint(1, 3)
            if have_key == 1:
                self.entities.append(Guard(pos))
            elif have_key == 2:
                self.entities.append(Mummy(pos))
            elif have_key == 3:
                self.entities.append(Pharaoh(pos))
            cnt += 1
        self.entities[random.randint(0, var.enemies_number -
1)].collectibles.append(Key())


    def add_door(self, to):
        """
        Добавляем дверь для комнаты. Вызывается в map.
        Портал добавляется в это же время.
        :param to: str. It only has four value: left, right, up, down. Represent which
door should be add.
        :return:
        """
        x1, y1 = DOOR_POS[to]
        x2, y2 = DOOR_POS[to][0] + PORTAL_POS[to][0], DOOR_POS[to][1] + PORTAL_POS[to][1]
        if to == 'left':
            closed_image = const.IMAGE['door'][2]
            opened_image = const.IMAGE['door'][3]
```

```
        elif to == 'right':
            closed_image = const.IMAGE['door'][4]
            opened_image = const.IMAGE['door'][5]
        else:
            closed_image = const.IMAGE['door'][0]
            opened_image = const.IMAGE['door'][1].copy()
            opened_image.blit(const.IMAGE['ground'][random.randint(0, 7)], (0, 24))
            opened_image.blit(const.IMAGE['door'][1], (0, 0))
        self.buildings[x1][y1] = Door((x1, y1), closed_image, opened_image)
        self.buildings[x2][y2] = Portal((x2, y2), to)

    def update(self):
        self.__update_entities()
        self.__update_collide()
        self.__update_trigger_buildings()

    def __update_entities(self):
        for entity in self.entities:
            entity.update()

    def __update_collide(self):
        """
        Сначала проверяем столкновение между сущностью и строением.
        Затем проверяем столкновение между сущностью и сущностью.
        :return:
        """
        # Сущность со зданиями.
        for entity in self.entities:
            buildings = []
            # Найдем все строения, столкнувшиеся с сущностью.
            _left = int(entity.rect.left) // TILE_SIZE[0]
            _right = int(entity.rect.right) // TILE_SIZE[0]
            _top = int(entity.rect.top) // TILE_SIZE[1]
            _bottom = int(entity.rect.bottom) // TILE_SIZE[1]
            # left_top
            buildings.append(self.buildings[_left][_top])
            # right_top
            if _left != _right:
                buildings.append(self.buildings[_right][_top])
            # left_bottom
            if _top != _bottom:
                buildings.append(self.buildings[_left][_bottom])
            # right_bottom
            if _left != _right and _top != _bottom:
                buildings.append(self.buildings[_right][_bottom])
            # Обработка столкновения.
            entity.collide_building(*buildings)
        # Сущность и сущность. Используем QuadTree.
        quadtree = QuadTree(Rect.init_two_arg((0, 0), self.__surface.get_size()))
        for entity in self.entities:
            quadtree.add(entity)
        for entity in self.entities:
            # Проверка, что существо будет другой сущностью.
            if not isinstance(entity, Creature):
                continue
            entities = []
            for collide_entity in quadtree.qry(entity):

                # Проверка, что столкновение объектов разного типа.
                # Монстры не сталкиваются с Collectible (и не могут их забрать).
                if type(collide_entity) == type(entity) or \
                        (isinstance(entity, Monster) and isinstance(collide_entity,
Collectible)):
                    continue
```

```
                if entity.rect.intersect(collide_entity.rect):
                    entities.append(collide_entity)
            # Обработка столкновения.
            entity.collide_entity(*entities)

    def __update_trigger_buildings(self):
        for trigger_building in self.__trigger_buildings:
            trigger_building.update()

    def draw(self, surface):
        """
        Используем bubble_sort для сортировки сущности в соответствии со значением
rect.bottom в порядке возрастания.
        :param surface: Surface.
        :return:
        """
        if not isinstance(surface, Surface):
            raise TypeError("Room.draw.surface must be Surface type.")
        # Пузырьковая сортировка.
        bubble_sort(self.entities, lambda entity1, entity2: entity1.rect.bottom -
entity2.rect.bottom)
        self.__surface.fill((0, 0, 0))
        cnt = 0
        for j in range(ROOM_SIZE[1]):
            # Отображаем строения.
            for i in range(ROOM_SIZE[0]):
                if self.buildings[i][j] != None:
                    self.buildings[i][j].draw(self.__surface)
            # Отображаем сущности.
            while cnt < len(self.entities) and self.entities[cnt].rect.bottom //
TILE_SIZE[1] == j:
                self.entities[cnt].draw(self.__surface)
                cnt += 1
        # Вычисляем точку (одна и та же в каждом кадре).
        self.__blit_point = ((surface.get_width() - self.__surface.get_width()) // 2,
                             (surface.get_height() - self.__surface.get_height()) // 2)
        surface.blit(self.__surface, self.__blit_point)
```

room_supply.py (пример комнаты с сундуком)

```
import random
import const
from src.obj.building.chest_supply import ChestSupply
from src.obj.building.wall import Wall
from src.room.room import Room

class RoomSupply(Room):
    def init_buildings(self):
        super().init_buildings()
        width = const.ROOM_SIZE[0] - 6
        height = const.ROOM_SIZE[1] - 6
        for i in range(width):
            for j in range(height):
                if ((i == width * 2 // 5 - 1 or i == width * 3 // 5) and
                    (height // 5 <= j <= height * 2 // 5 or height * 3 // 5 <= j <=height
* 4 // 5)) or \
                        ((j == height * 2 // 5 or j == height * 3 // 5) and
                         (width // 5 - 1 <= i <= width * 2 // 5 - 1 or width * 3 // 5 +
1<= i <= width * 4 // 5 )):
                    self.buildings[i + 3][j + 3] = Wall((i + 3, j + 3),
const.IMAGE['wall'][5], (0, -2))
        self.buildings[const.ROOM_SIZE[0] // 2][const.ROOM_SIZE[1] // 2 - 1] = \
            ChestSupply((const.ROOM_SIZE[0] // 2, const.ROOM_SIZE[1] // 2 - 1))
```

chest_supply.py (пример сундука с расходниками)

```
import random
import var
from const import IMAGE
from src.obj.building.trigger_building import TriggerBuilding
from src.tool.vector import Vector
from src.obj.entity.item.bullet import Bullet
from src.obj.entity.item.potion_hp import PotionHP
from src.obj.entity.item.potion_buff import PotionBuff
from src.obj.entity.item.key import Key
"""
Сундук с боеприпасами, лечением и ключом (ключами)
"""
class ChestSupply(TriggerBuilding):
    def __init__(self, pos, *collectibles):
        super().__init__(pos, IMAGE['chest'][4], (0, 0), False, False, IMAGE['chest'][5],
(0, 0))
        self.__collectibles = [*collectibles]
        self.__collectibles = [*collectibles]
        # Добавляем выпадение боеприпасов, зелий и ключей
        self.__collectibles.append(Bullet(random.randint(2, 3) * 10))
        self.__collectibles.append(PotionHP(random.randint(1, 3)))
        self.__collectibles.append(PotionBuff(random.randint(1, 3)))
        self.__collectibles.append(Key(random.randint(1, 2)))
    @property
    def collectibles(self):
        return self.__collectibles

    def on_trigger(self, entity):
        if entity != var.player:
            return
        if self.can_trigger is True:
            self.image = self.new_image
            self.vector = self.new_vector
            for collectible in self.__collectibles:
                collectible.explode(self.rect.center, Vector.random_normalized_vector())
                var.map.active_room.entities.append(collectible)
            self.can_trigger = False
```

trap.py

```
import const
import var
from src.animation.animation_system import AnimationSystem
from src.debuff.debuff_slow import DebuffSlow
from src.obj.building.trigger_building import TriggerBuilding
"""
Это единственное строение, у которого есть анимация.
Когда ловушка показывает колючки, то наносит урон игроку и накладывает на него дебафф
замедления, если игрок наступит на него.
Ловушки никак не взаимодействуют с монстрами.
"""
class Trap(TriggerBuilding):
    def __init__(self, pos):
        super().__init__(pos, None, (0, 0), True, False, None, (0, 0))
        self.__animation_system =
AnimationSystem(**const.ANIMATION_REPOSITORY.animations['trap'])
        self.__animation_system.play('work')

    def update(self):
        self.__animation_system.update()
        self.image = self.__animation_system.image

    def on_trigger(self, entity):
        # Only player get hurt.
        if entity != var.player or self.image != const.IMAGE['trap'][3]:
            return
        var.player.take_damage(2)
        var.player.debuff = DebuffSlow(180, var.player, -2)
```

debuff_slow.py

```
from src.debuff.debuff import Debuff
"""
Дебафф, который заставит игрока двигаться медленно на некоторое время.
Этот класс также используется в качестве ускорения, которое дает зелье баффа.
"""
class DebuffSlow(Debuff):
    def __init__(self, time, creature, level=1):
        super().__init__("Changed (" + str(level) + " points)", time, creature)
        self.level = level

    @property
    def level(self):
        return self.__level

    @level.setter
    def level(self, value):
        if not isinstance(value, int):
            raise TypeError("DebuffSlow.level must be int type.")
        elif not -3 <= value <= 4:
            raise ValueError("DebuffSlow.level should be -3 to 4.")
        self.__level = value

    def take_effect(self):
        self.creature.speed_mag *= 1 + 0.2 * self.level

    def recover(self):
        self.creature.speed_mag /= 1 + 0.2 * self.level
```

debuff.py

```
"""
Экземпляр дебаффа есть только у существа.
Это абстрактный базовый класс.
update() вызывается каждый кадр.
"""
class Debuff:
    def __init__(self, name, time, creature=None):
        self.__name = name
        self.time = time
        self.__creature = creature
        self.__flag = True

    @property
    def name(self):
        return self.__name

    @property
    def time(self):
        return self.__time

    @time.setter
    def time(self, value):
        self.__time = int(value)

    @property
    def creature(self):
        return self.__creature

    def update(self):
        if self.creature == None or self.time == 0:
            return
        if self.__flag:
            self.__flag = False
            self.take_effect()
        self.time -= 1

    def take_effect(self):
        pass

    def recover(self):
        pass
```

food_buff.py (реализация баффа)

```
import var
from src.debuff.debuff import Debuff
from src.debuff.debuff_slow import DebuffSlow
from src.obj.entity.item.collectible import Collectible


class FoodBuff(Collectible):
    def __init__(self, name, image, amount):
        super().__init__(name, image, True, amount)

    @property
    def can_use(self):
        return (var.player.debuff != None and var.player.debuff.name == "Changed (-2
points)" or var.player.debuff == None)


    @can_use.setter
    def can_use(self, value):
        pass

    @property
    def debuff(self):
        return self.__debuff

    @debuff.setter
    def debuff(self, value):
        if value != None and not isinstance(value, Debuff):
            raise TypeError("Creature.debuff must be Debuff or None type.")
        if value != None and var.player.debuff != None:
            return
        self.__debuff = value

    def use(self):
        if var.player.debuff != None:
            if var.player.debuff.name == "Changed (3 points)":
                return
            if var.player.debuff.name == "Changed (-2 points)":
                var.player.debuff.recover()
                var.player.debuff = None
                super().use()
                return
        super().use()
        var.player.debuff = DebuffSlow(600, var.player, 3)
```

potion_buff.py (зелье баффа)

```
from src.obj.entity.item.food_buff import FoodBuff
from const import IMAGE
"""
При его использовании скорость игрока увеличится, негативные эффекты снимутся.
Зелье не может быть использовано, если игрок уже под действием баффа.
"""
class PotionBuff(FoodBuff):
    def __init__(self, amount=1):
        super().__init__("Buff potion", IMAGE['potion'][1], amount)

    @property
    def information(self):
        return '    This buff potion makes you move faster for a while.'
```

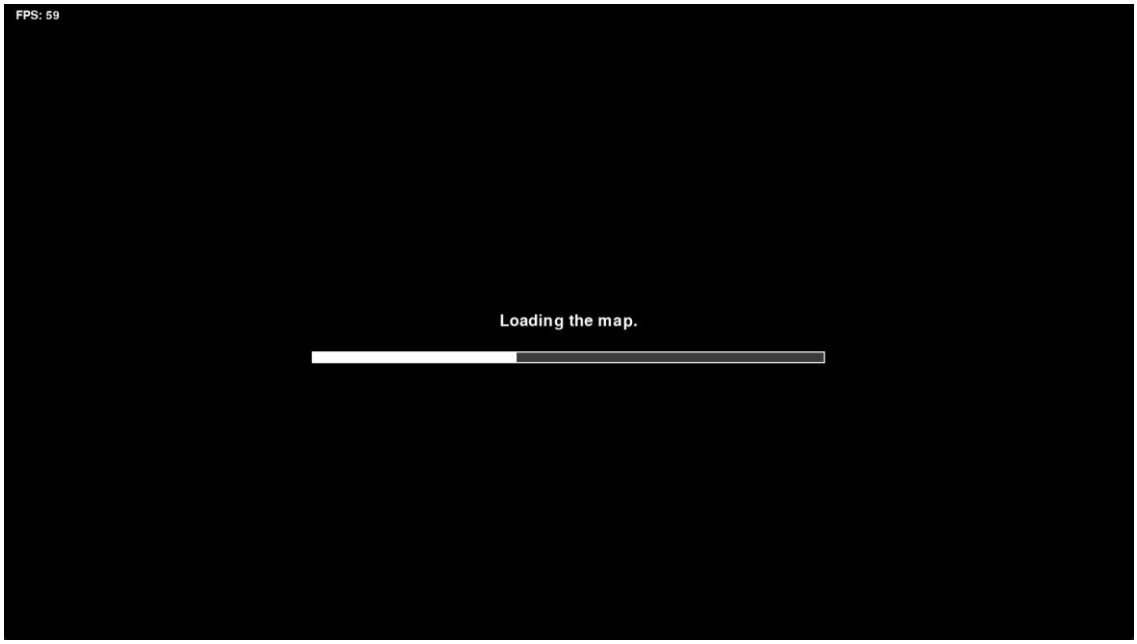
food_hp.py (реализация лечения)

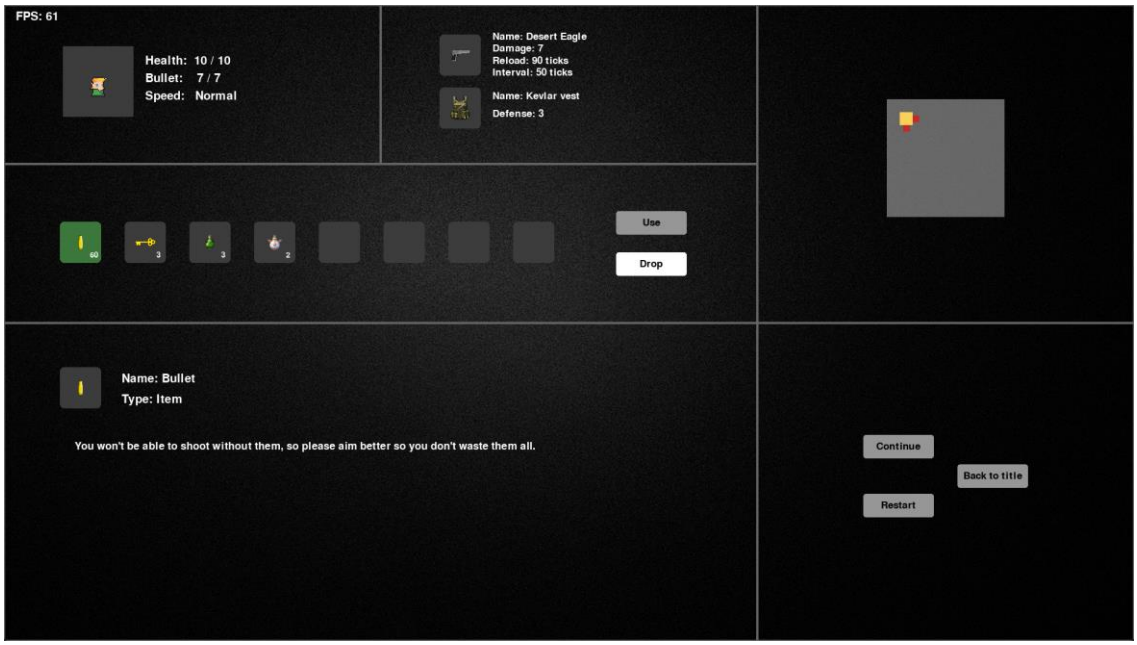
```
import var
from src.obj.entity.item.collectible import Collectible


class FoodHP(Collectible):
    def __init__(self, name, image, amount, recover_health):
        super().__init__(name, image, True, amount)
        self.recover_health = recover_health

    @property
    def can_use(self):
        return var.player.health != var.player.max_health

    @can_use.setter
    def can_use(self, value):
        pass

    @property
    def recover_health(self):
        return self.__recover_health

    @recover_health.setter
    def recover_health(self, value):
        self.__recover_health = int(value)

    def use(self):
        if var.player.health == var.player.max_health:
            return
        super().use()
        var.player.health += self.recover_health
```

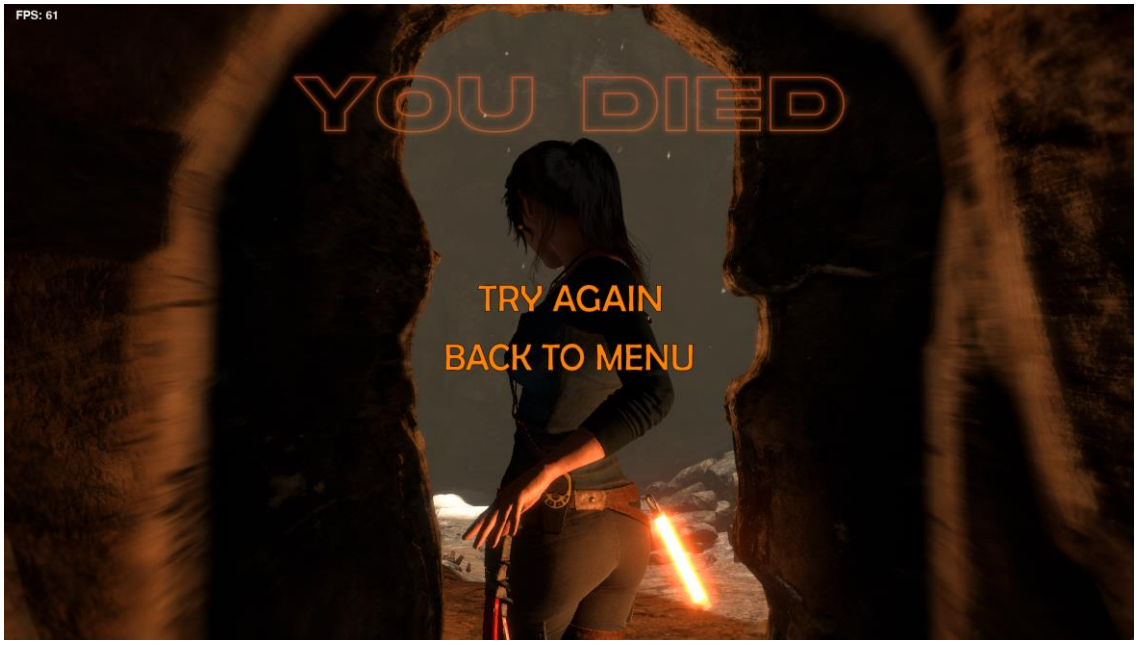
potion_hp.py (зелье лечения)

```
from src.obj.entity.item.food_hp import FoodHP
from const import IMAGE
"""
При его использовании здоровье игрока увеличится.
Зелье не может быть использовано, если игрок полностью здоров.
"""

class PotionHP(FoodHP):
    def __init__(self, amount=1):
        super().__init__("Healing potion", IMAGE['potion'][0], amount, 3)

    @property
    def information(self):
        return '    This healing potion can restore 3 points of your HP.'
```

Результат работы программы




FPS: 61
Health: 10 / 10
Bullet: 7 / 7
Speed: Normal
Name: Desert Eagle
Damage: 7
Reload: 90 ticks
Interval: 50 ticks
Name: Kevlar vest
Defense: 3
Use
Drop
Name: Bullet
Type: Item
You won't be able to shoot without them, so please aim better so you don't waste them all.
Continue
Back to title
Restart


FPS: 61
YOU DIED
TRY AGAIN
BACK TO MENU


FPS: 61
DESIGNED AND DIRECTED BY KEEPREVERSE
BACK TO MENU

Код программы

Код программы слишком большой, чтобы полностью указать его в документе, поэтому я оставлю ссылку на мой github репозиторий проекта игры: https://github.com/keepreverse/TombRaider-TheOrigins

Список источников

1. https://www.python.org
2. https://github.com/pygame/pygame
3. https://habr.com/ru/post/193888
4. https://github.com/Rion5/Pygame_2D_Game
5. https://github.com/ArminBolic/Hero-Quest