МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«Южный федеральный университет»

Институт математики, механики
и компьютерных наук им. И. И. Воровича

Забродина Елизавета Михайловна
Красин Александр Сергеевич
Лаврикова Елена Сергеевна
Точилин Никита Андреевич

КОМПБЮТЕРНАЯ ИГРА НА ТЕМУ БЕЗОПАСНОСТИ В ИНТЕРНЕТЕ

ПРОЕКТ 2 КУРСА
по направлению подготовки
01.03.02 – Прикладная математика и информатика

Научный руководитель –
доцент Пустовалова Ольга Геннадиевна

оценка (рейтинг) подпись руководителя

Ростов-на-Дону – 2023

Оглавление

Постановка задачи

1) Написание компьютерной программы на языке Python с реализацией функция работы с графикой, звуком, вводом данных с клавиатуры.

2) Поиск информации по теме, оформление в форме коротких сообщений.

3) Создание графических и звуковых ресурсов для использования в игре.

4) Разработка уровней игры

Введение

Для написания программы был использован язык Python с подключением стандартной библиотеки System, а так же сторонней библиотеки Pygame. Эта библиотека включает в себя весь необходимый функционал для работы графикой звуком и вводом данных с клавиатуры.

При разработке программы в приоритет был поставлен принцип модульности. Код программы не содержит описания уровней и информационных материалов. Вместо этого нами были написаны функции для загрузки и обработки информации из сторонних файлов. Благодаря этому исполняемый файл не содержит лишних данных, а добавление новых уровней/записей не требует внесения изменений в код.

Все ресурсы программы структурированы по файлам. Это облегчает процесс разработки и навигации по файлам, а также снижает вероятность возникновения конфликтов имён. Аналогично структурирован по файлам и код программы. Исполняемая часть кода находится в файле *Main.py*. Описания классов и функций, инициализация констант, а также загрузка и инициализация графических и звуковых ресурсов происходит в отдельных модулях, подключаемых в основном файле.

В программе активно применяются принципы ООП. Нами был создан набор классов, отвечающих за различные объекты игрового мира, а так же классы-контроллеры для удобной реализации отдельных элементов программы.

Визуальная составляющая выполнена при помощи растровой графики. Она состоит из графических примитивов, сгенерированных и отрисованных при помощи библиотеки PyGame, а также набора изображений(спрайтов) созданных отдельно в графическом ПО и загружаемых в программу.

Файловая Структура

Помимо исполняемого файла, в директории игры так же храняться ресурсы игры и сохранённые данные. Все они разбиты по соответствующим файлам.

- levels - уровни
- resources - ресурсы
- saves - сохранения
- codex - кодекс

Уровни: Директория *levels* содержит все уровни игры в виде папок, а так же файл *levels.txt*. Этот файл содержит имена всех уровней. Каждая строка файла содержит имя одного уровня. Порядок записи уровней в этом файле определяет порядковый номер(индекс) уровня внутри программы. Такой способ загрузки гарантирует, что уровень получит нужный порядковый номер вне зависимости от своего наименования и метода сортировки файлов в ОС.

Уровень состоит из n-ого количества этапов. Каждый этап представляет является законченным игровым пространством и содержит описание расположения всех объектов на нём.

Папка уровня имеет структуру, аналогичную ранее описанной: некоторое количество папок этапов, а так же связывающий их файл *stages.txt*. Файл *stages.txt*, по аналогии с файлом *levels.txt* содержит список имён этапов, который определяет присвоение им порядковых номеров.

Этап: Папка этапа содержит набор файлов формата *txt*. Отсутствие или неправильное оформление любого из этих файлов при запуске программы вызовет ошибку "повреждение файлов".

Структура этапа:

- i*info.txt* - файл содержащий основные данные об уровне, а именно:
  - **Координаты старта** в формате "x y два числа типа integer записываются через пробел. На эти координаты будет перемещён

игрок при запуске/перезапуске этапа.

– -**Фоновая музыка** - string без пробелов. Имя аудиофайла, которые будет проигрываться фоном во время этапа. Сам файл хранится в соответствующей папке в разделе "ресурсы".

Программа разбивает строки файла на слова и читает только нужное ей количество слов. Для строки координат берётся 2 слова, для музыки 1 слово. Все последующие символы не учитываются и могут быть использованы для комментирования.

Пример файла :

```
2 2 #Координаты
mymusic.mp3 #Музыка
```

- *layout.txt* - расположение стен. Стены это твёрдые объекты, через который не может пройти игрок или другой персонаж, управляемый компьютером. Их расположение задаётся схематически в файле *layout.txt*. Количество символов с строке не должно превышать 36, а количество строк 22. Каждый символ этого файла отражает одну ячейку игрового мира. Все символы, отличные от пробела, означают стены. При этом разные символы могут обозначать разные виды стен(отличие исключительно визуальное, все стены имеют одинаковую механику поведения). Виды стен заданы в методе *read_stage_data* класса *LevelStageSetup*. В этом методе задано n видов стен, все символы не описанные в методе(кроме пробела) будут интерпретированы как стандартная стена.

  Комментарии или любые другие данные в этом файле не допускаются.

Пример файла *layout.txt*:

```
###########
#         #
#         #
#    #    #
#    #    #
###########
```

Для удобного создания уровней нами был использован инструмент ASCII_Art_Paint.

- Файлы *bonuses_list.txt*, *doors_list.txt*, *enemies_list.txt*, *portals_list.txt* и *codex.txt* содержат информацию об объектах на данном этапе. Эти файлы имеют аналогичную структуру. Каждый из них содержит n строк. Каждая строка отвечает за отдельный объект и содержит его характеристики. Строки начинающиеся с символа # не читаются программой и нужны для комментирования.
  Для каждого типа объектов свой набор характеристик. Общими для всех являются координаты x y записанные в форме двух числел типа integer.
  - Файл *doors_list.txt* описывает двери и содержит только координаты.
    Пример:

    ```
    #x y
    2 5
    3 8
    9 24
    ```

  - Файл *codex.txt* описывает записки и содержит координаты, а также идентификатор открываемой записи кодекса в формате строки string без пробелов.
    Пример:

```
#x y id
9 9 e_1_1
13 27 e_1_2
```

– Файл *bonuses_ list.txt* описывает бонусы и содержит их координаты, идентификатор отображаемого спрайта в формате строки без пробелов ("heal"или "key идентификатор является ключём к словарю спрайтов объекта типа бонус и может быть расширен. В данной версии игры применяются эти два значения), а так же 4 числа integer отвечающих за то, сколько данный бонус добавит игроку ключей, жизней, очков и с какой частотой будет отображаться анимация спрайта.

Пример:

```
#x y spr_id keys hp score anim_del
#Ключи
25 2 key 1 0 100 60
29 2 key 1 0 100 60
#
#Жизни
6 6 heal 0 1 200 60
10 15 heal 0 1 200 60
```

– Файл *enemies_ list.txt* описывает врагов(вирусы) на данном этапе и содержит их координаты, вектор движения по двум координатам, идентификатор спрайта, скорость движения и частоту анимации в виде чисел типа integer.

Пример:

```
#x y v_x v_y spr_id spd anim_delay
10 4 0 1 0 2 60
14 4 0 -1 0 2 60
19 1 1 0 0 2 60
```

– Файл *portals_list.txt* описывает порталы, переносящие игрока на другие этапы и уровни. Он содержит координаты объекта, порядковые номера уровня и этапа, на которые должен перейти игрок и новые координаты игрока в виде чисел integer.
Пример:

```
#x y lvl stg p_x p_y delay
33 19 1 0 2 2 60
```

Ресурсы: Директория *resources* содержит графические и аудио ресурсы игры. Они разбиты по разделам *sprites* - графика, *music* - фоновая музыка и *sound* - звуки.

Сохранения: Директория *saves* содержит 2 файла сохранений.

- Файл *player_save.txt* хранит информацию о прогрессе игрока, а именно порядковые номера уровня и этапа(1 строка), координаты игрока(2 строка), количество жизней(3 строка), количество набранных очков(4 строка).
Пример:

```
1 1
64 720
3
4100
```

- Файл *codex_save.txt* хранит информацию об открытых записях кодекса в формате идентификатор:значение, значение выражается цифрой 1 или 0.

Пример:

```
e_start:1
e_0_0:1
e_3_0:1
e_3_1:1
```

Кодекс: Директория *codex* содержит записи, открываемые в процессе игры. Каждая запись представлена отдельным файлом формата *.txt*. Первая строка файла отвечает за видимость записи по умолчанию. 1 означает, что запись видна игроку изначально, 0 - запись требуется открыть. Вторая строка задаёт уникальный идентификатор записи в форме строки string без пробелов. Последущие строки отображаются в виде текста при просмотре записи в игре.

Пример:

```
0
e_3_2
Внимательно выбирайте интернет-магазины
Перед совершением сделки убедитесь, что
перед вами не сайт-однодневка и проверьте
отзывы о магазине. В идеале пользуйтесь
услугами крупных торговых площадок.
Пользуйтесь оплатой через банковские сервисы
не переводите деньги продавцу на личный счёт.
```

Алгоритм работы программы

Программа написана на языке python с применением модуля PyGame. Данный модуль предполагает покадровую обработку игры. После запуска программы сначала выполняется сегмент инициализации в котором подключаются дополнительные модули, загружаются все игровые ресурсы,

инициализируются переменные и создаются игровые объекты. После этапа инициализации начинается игровой цикл. Одна итерация цикла обрабатывает один кадр игры. Цикл продолжает работу до тех пор, пока переменная *game_ cycle* не примет значение *False*.

Структура файла *main.py*:

```
import pygame
...
<подключение дополнительных модулей>
...
<инициализация переменных>
...
while game_cycle:
    clock.tick(60) #Ограничение частоты кадров в секунду на 60
    ...
    <обработка кадра>
```

Алгоритм работы игры описан при помощи набора игровых этапов (не путать с этапами уровня). Переменная *game_ stage* хранит номер текущего этапа игры в виде числа integer. Каждый кадр в начале цикла программа проверяет номер этапа и запускает соответствующую секцию кода. Выбор этапа реализован с помощью структуры *match-case*. Перед ней идёт стандартный для програм, написанных с использованием PyGame, обработчик событий, который считывает нажатия кнопок клавиатуры и закрытие окна игры.

Структура итерации игрового цикла:

```
while game_cycle:
    clock.tick(60)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            game_cycle = False #Завершение игры по нажатию крестик
     pressed = pygame.key.get_pressed() #переменная pressed хранит

     match game_stage:
        case -1:
            ...
            <экран ошибки>
            ...
        case 0:
            ...
            <главное меню>
            ...
        case 1:
            ...
            <меню кодекса>
            ...
        case 2:
            ...
            <список авторов>
            ...
        case 3:
            ...
            <меню выбора режима игры>
            ...
        case 10:
            ...
```

```
        <игровой процесс>
        ...
    case 11:
        ...
        <экран победы>
        ...
```

Каждая секция в свою очередь имеет следующую стрктуру

1) Заливка экрана фоновым цветом;

2) Обработка объектов и их взаимодействия;

3) Отрисовка объектов

4) Вывод изображения на экран

Классы и ООП в проекте

За исключением нескольких глобальных переменных, все остальные данные хранятся в объектах - экземплярах различных классов. Проект не содержит ни одной независимой функции. Все функции выполнены в виде методов классов.

Классы в проекте можно условно разделить на классы игровых объектов и классы-контроллеры.

- У классов игровых объектов есть множество экземпляров, каждый из которых отвечает за отдельный объект(например бонус, вирус или запись в кодексе)
- Классы-контроллеры имеют только один экземпляр в реализации программы и используются для удобного и структурированного управления отдельными системами игры(например контроллер Кодекса, контроллер Уровня)

В некотором смысле исключением является класс *player*. С одной стороны этот класс отвечает за объект игрока, с другой стороны именно

в этом классе реализованы глобальные системы сохранения/загрузки и он имеет лишь один экземпляр, что свойственно Контроллерам.

Классы

StaticObject отвечает за статичные игровые объекты. К таким, например, относятся бонусы и двери.

Поля класса:

- **pos_x** - координата x
- **pos_y** - координата y
- **col_rect** прямоугольник столкновений, он же "хит-бокс". С помощью этого прямоугольника происходит обработка столкновения объектов
- **sprite** - набор изображений(может состоять из одного изображения) который используется для отображения объекта.
- **delay** - задержка между кадрами анимации спрайта(например значение 60 означает, что кадр анимации будет переключаться каждый 60 кадр игры)

Методы класса:

- **collide_get_id** - получает список любых объектов и возвращает номер объекта с которым данный пересекается.
- **change_sprite** - заменяет спрайт объекта
- **draw_stat** - отрисовывает спрайт без анимации. По умолчанию используется первый кадр спрайта, можно выбрать любой кадр по номеру доп. аргументом *spr_id*
- **draw_anim** - отрисовывает спрайт с анимацией

DynamicObject (наследник *StaticObject*) отвечает за динамические объекты.

Поля класса:

- все поля родительского класса
- **speed** - скорость движения

- **v_x** - значение x вектора движения объекта
- **v_y** - значение y вектора движения объекта

Методы класса:

- все методы родительского класса
- **coords** - возвращает координаты объекта в виде пары

Player (наследник *DynamicObject*) - объект игрока.

Поля класса:

- все поля родительского класса
- **hp** - количество жизней игрока
- **keys** - количество собранных ключей
- **score** - количество набранных очков
- **direction** направление движения игрока, выраженное в виде целого числа на отрезке [0,3], где 0 - вверх, 1 - вниз, 2 - право, 3 - лево.
- **hard_mode** - флаг(bool) усложнённого режима игры

Методы класса:

- все методы родительского класса
- **read_controls** - перемещает игрока по игровому полю в соответствии с нажатыми клавишами.
- **add_hp, add_keys, add_score** - три метода изменения соответствующих полей объекта "игрок". В методы уже встроена проверка на выход за границы допустимых значений полей(так счёт не может быть меньше нуля, а количество жизней превышать заданный максимум).
- **save** - сохранение игры. Принимает в качестве аргументов номера текущего уровня и этапа, так как эти данные не содержаться в экземпляре класса игрок.
- **load** - загрузка данных из файла сохранения. Записывает полученные данные в объект класса игрок, возвращает список из 2 элементов - номеров уровня и этапа.

Enemy (наследник *DynamicObject*) - объект противника(вируса). Поля и методы данного класса полностью совпадают с полями и методами родительского класса.

Bonus (наследник *StaticicObject*) - объект бонуса(пополнения жизни или ключа)

Поля класса:

- все поля родительского класса
- **add_hp** - количество жизней, которое получит игрок при взятии бонуса.
- **add_keys** - количество ключей, которое получит игрок при взятии бонуса.
- **score** - количество очков, которое получит игрок при взятии бонуса.

Все методы класса совпадают с методами родительского класса.

CodexNote (наследник *StaticicObject*) - "записка"взяв которую игрок открывает запись в кодексе.

Поля класса:

- все поля родительского класса
- **codex_id** - идентификатор записи, которая откроется в кодексе, при взятии записки.

Все методы класса совпадают с методами родительского класса.

Portal (наследник *StaticicObject*) - объект "портал"перемещающий игрока на другой этап или уровень.

Поля класса:

- все поля родительского класса
- **new_lvl, new_stg** - два поля, хранящие номера уровня и этапа, на которые перенесёт игрока.
- **new_player_pos** - список из двух элементов. Новые координаты игрока, после перемещения.

Все методы класса совпадают с методами родительского класса

LevelStageSetup - контроллер загрузки объектов и данных этапа уровня. Является важнейшим контроллером для работы игры.

Поля класса:

- **start_pos** - список из 2 элементов. Начальные координаты игрока, присваиваемые при загрузке уровня.
- **music_path** - строка, путь к файлу фоновой музыки этапа.
- **walls, enemies, bonuses, doors, portals, codex_notes** - списки, содержащие все объекты этапа, сгруппированные по типу(стены, враги, бонусы, двери, порталы, записки).

Методы класса:

- **read_stage_data** - метод загрузки всех данных этапа. В качестве аргументов метод принимает две строки - имя папки уровня и имя папки этапа. Данные считываются из файлов директории этапа, структура этих файлов была ранее описана в разделе *"Файловая структура"*.

  Каждая группа объектов загружается отдельно с помощью цикла, проходящего по соотвествующему файлу. Каждый такой цикл имеет аналогичную структуру с поправкой на необходимый для данного типа объектов набор параметров.

Структура цикла:

```
with open(<путь к файлу>) as f:
    ss = f.readlines() #Разбиваем файл на строки
    for s in ss:
        if s[0] != "#": #Проверка на комментарий
            _ss = s.split() #разбиваем строку
            arg1 = _ss[0]
            arg2 = _ss[1]
            ...
            argn = _ss[n-1]
            self.obj_list.append(Object(arg1, ... argn))
```

В данном примере мы считали из файла данные об объектах класса Object, конструктор которого принимает n аргументов и записали их в список obj_list. На месте абстрактных Object и obj_list могут быть любые ранее описанные классы, списки которых реализованы в данном контроллере.

image_display - графический объект. В отличии от класса *StaticObject* не имеет прямоугольника столкновений и, как следствие, не может взаимодействовать с другими объектами. Нужен для отрисовки графических элементов на экране.

Поля класса:

- **x, y** - два поля, координаты объекта.
- **sprite** - спрайт, отображаемый при отрисовке объекта.

Методы класса:

- **draw** - отрисовывает объект на экран.

text_display - текстовый графический объект. Нужен для отрисовки текста на экран.

Поля класса:

- **x, y** - два поля, координаты объекта.
- **text** - отображаемый текст.
- **font** - шрифт отображаемого текста. Шрифты инициализируются при запуске программы, аргументом передаётся экземпляр класса font из модуля Pygame.
- **color** - цвет отображаемого текста.

Методы класса:

- **draw** - отрисовывает объект на экран.

MessageBox - объект всплывающего текстового окна. Является вспомогательным классом и не используется вне класса-контроллера *MessageBoxContr*

Поля класса:

- **text** - отображаемый текст.
- **font** - шрифт отображаемого текста.
- **color** - цвет окна.
- **font_color** - цвет текста.
- **duration** - длительность отображения окна на экране в кадрах. Например *duration=180* означает, что окно будет оставаться на экране 180 обновлений кадров, т.е. примерно 3 секунды при правильной работе программы.
- **height, width** - высота и ширина окна. Задаются автоматически в конструкторе класса исходя из количества символов *text* и размера шрифта *font*.

Методы класса:

- **show** - выводит окно в середину экрана.

MessageBoxController - контроллер системы всплывающих окон. экземпляр класса хранит в себе очередь сообщений и отвечает за их последовательный показ.

Поля класса:

- **message_list** - список, очередь сообщений, которые нужно вывести на экран.

Методы класса:

- **add_message_box** - добавляет переданный в аргументе объект класса *MessageBox* в очередь.
- **update_message_box** - обновляет статус контроллера. Данный метод должен вызываться на каждом кадре игрового процесса. Если в очереди есть хотя бы одно сообщение, первое из них выводится на экран и его значение *duartion* уменьшается на 1. Если значение *duartion* первого элемента равняется 0, этот элемент удаляется из очереди. Этот алгоритм продолжает работу, пока есть хотя бы один элемент в очереди. Если очередь пуста, то контроллер будет ожидать новых элементов.

MenuController - контроллер управления меню. Представляет из себя комплекс объектов для отрисовки вертикального меню с любым количеством опций.

Поля класса:

- **x,y** - координаты блока меню на экране.
- **option_list** - список идентификаторов опций меню. Все элементы данного списка должны быть уникальны. Именно эти идентификаторы используются программой для обработки выбора пользователя.
- **names_list** - список отображаемых имён опций. Длина этого списка должна быть равна длинне списка *option_list*. Именно эти названия выводятся на экран.
- **cursor_pos** - положение курсора в списке. Целое число [0, n-1], где n - количество опций.
- **cooldown** - задержка после нажатия клавиши. Пока это значение не равно 0, ввод данных с клавиатуры будет игнорироваться. Это значение повышается после каждого нажатия и понижается каж-

дый кадр без нажатий. Эта задержка нужна для точного управления меню.

Методы класса:

- **read_controls** - по функционалу схож с одноимённым методом класса **Player**. Принимает ввод клавиш с клавиатуры и изменяет положение курсора в меню.
- **update** - покадровое обновление меню. Каждый кадр курсор(объект класса *image_display* отрисовывается напротив выбранного пункта меню. Так же именно этот метод отвечает за обработку задержки ввода.
- **get_options** - возвращает идентификатор опции меню, на который в данный момент направлен курсор.

CodexEntry - класс отдельной записи кодекса. По аналогии с классом *MessageBox*, данный класс является вспомогательным для класса-контроллера

Поля класса:

- **visivble** - флаг(bool) видимости записи. Если значение флаг = False, то содержание записи не будет отображаться.
- **id** - идентификатор записи. У каждой он должен быть уникален.
- **lines** - список строк, текст записи.

Методы класса:

- **unlock** - метод обращения к полю *visible*. Меняет его значение на True. Обращение значения на False в процессе работы программы не предусмотрено. Однажды открытая запись должна оставаться такой.

Codex_Controller - контроллер Кодекса.

Поля класса:

- **n** - номер отображаемой на экране записи. Не путать с идентификатором. Номер - целое число, номер элемента в списке *entries*

данного класса. Идентификатор - строка, по которой можно определить объект класса *CodexEntry*.

- **cooldown** - задержка, аналогично классу **MenuController**
- **entries** - список всех записей в кодексе(объектов класса *CodexEntry*.

Методы класса:

- **entry_status** - принимает идентификатор и возвращает логическое значение(bool) записи с этим идентификатором из списка *entries*.
- **unlock** - вызывает метод *unlock* для записи с заданным идентификатором. Иными словами разблокирует запись из списка по её идентификатору.
- **show_entry** - выводит на экран запись под номером n, если запись разблокирована, либо текст

  ```
  Запись заблокирована
  Найдите файл на уровне игры, чтобы её разблокировать
  ```

  Если эта запись заблокирована.
- **read_controls** - управление при помощи клавиатуры. Аналогично одноимённым методам других классов.
- **get_from_files** - читает записи из папки *codex* и заполняет ими список *entries*. Вызывается в начале работы программы сразу после инициализации контроллера.
- **save** - записывает в файл *codex_save.txt* идентификаторы *id* и статус *visible* всех записей.
- **load** - читает файл *codex_save.txt* и обновляет статусы всех записей в соответствии с полученной информацией.

Игровой Процесс

Запустив игру, пользователь попадает в главное меню, реализованное классом *MenuController*. При помощи клавиш **W** и **S** клавиатуры он может перемещаться по пунктам меню. С помощью клавиши **Пробел** он может подтвердить свой выбор. Главное меню состоит из 4 пунктов:

- **Игра** - выбрав этот пункт пользователь перейдёт к меню выбора режима игры/загрузки сохранения.
- **Кодекс** - этот раздел позволяет игроку в любое время просматривать записи, открытые в процессе игры. На экране кодекса пользователь может переключаться между записями с помощью клавиш **A** и **D**, пролистывая влево и вправо соответственно. Нажатие клавиши **esc** вернёт пользователя в главное меню, о чём сообщает надпись в правом верхнем углу экрана.
- **Авторы** - данный экран содержит имена всех авторов игры. Вернуться на главный экран также можно при помощи клавиши **esc**.
- **Выход** - выбор данной опции присвоит переменной *game_cycle* значение False и прервёт игровой цикл, что приведёт к завершению работы программы.

Выбрав опцию *Игра* пользователь перейдёт в меню выбора режима игры. Схема управления аналогична главному меню. Меню состоит из 4 опций:

- **Новая Игра(Легко)** - Запустить первый этап первого уровня в лёгком режиме.
- **Новая Игра(Сложно)** - Запустить первый этап первого уровня в сложном режиме.
- **Загрузить** - Происходит попытка загрузить файл сохранения *player_save*. Если это удаётся, то игрок продолжает игру с места сохранения. Если файла не существует, игра начинается с начала в лёгком режиме.
- **Назад** - Возвращает пользователя на главное меню.

Загрузив этап, персонаж, управляемый игроком, оказывается на двухмерном игровом поле в некоторой стартовой точке. Игрок видит сразу весь этап. Камера статична. Этап представляет из себя закрытое пространство, окружённое стенами и разбитое на отдельные помещения. Игрок может перемещать персонажа с помощью ввода с клавиатуры. Для этого используются клавиши **W,A,S,D**. Дополнительно клавиша **esc** возвращает игрока в главное меню.

Задача игрока состоит в безопасном перемещении персонажа на точку выхода с этапа, обозначенную оранжевым символом двери. Персонаж не может проходить сквозь стены. Персонаж не может проходить сквозь двери, но может открывать их. Для этого игроку нужен хотя бы один ключ. Если персонаж подойдёт вплотную к закрытой двери, то она откроется(с технической точки зрения удалится из этапа), при этом количество ключей у игрока уменьшится на 1. Ключи можно собирать на уровне. При контакте персонажа с бонусом, обозначенным символом ключа, бонус исчезает, а количество ключей у игрока увеличивается.

Персонаж начинает игру с 3 жизнями. Их общее количество не может превышать 6. Количество жизней может быть увеличено при взятии бонуса, обозначенного квадратом с крестиком внутри. Однако, если текущее количество жизней равно максимуму, бонус исчезнет, а дополнительная жизнь не будет добавлена.

Этап игры населяют противники - вирусы. Они перемещаются по простой траектории. Вирус двигается в одном направлении и, достигнув препятствия, меняет направление движения на противоположное. Скорость движения противника постоянная, но индивидуальная для каждого объекта. При контакте с персонажем, вирус исчезает, а игрок теряет одну жизнь.

Если количество жизней доходит до нуля, то игра считается проигранной. При этом на лёгком режиме сложности игрок начинает заново только текущий этап. Количество жизней возвращается на значение три, а на собранные очки накладывается штраф в 500 очков. Количество очков не может быть отрицательно. Если у игрока менее 500 очков и на него накладывается штраф, то количество очков обращается в ноль.

На сложном режиме, игроку даётся лишь один шанс на прохождение игры. Если количество жизней опускается до нуля, то игра начинается заново с первого этапа первого уровня. При этом делается сохранение с новой информацией о прогрессе. Эта мера исключает возможность возврата к прошлой точке сохранения в случае проигрыша.

На каждом этапе расположены 3 записки. Каждая из них открывает запись в кодексе. Однажды собранная записка больше не появляется на

этапе, даже если начать новую игроку. Механика их появления зависит от данных класса *Codex_Controller* и файла сохранения *codex_save.txt* и не зависит от данных файла *player_save.txt*. Записки расположены в труднодоступных местах, как правило за дверями или некоторыми опасностями. Это побуждает игрока рисковать, ради открытия записей и создаёт интересный игровой процесс.

На каждом этапе присутствует ровно один портал выхода, обозначенный изображением двери. Контакт с ним переносит игрока на следующий этап. При перемещении игра автоматически делает сохранение прогресса. Но, как было обазначенно ранее, проигрыш на высокой сложности перезаписывает это сохранение.

Портал на последнем этапе игры завершает её и включает экран победы. На нём игроку сообщается, что игра пройдена и выводится его итоговый счёт. Сообщение внизу экрана предлагает вернуться на главный экран. После завершения, игрок может ознакомится с записями в Кодексе или начать игру с начала.

Графика

Графика в игре реализована в виде растровых изображений. Все объекты игрового поля имеют размер 32 на 32 пикселя. Соответственно само игровое поле условно разбито на клетки 32 на 32 пикселя. Статичные объекты строго зафиксированны по сетке, динамические же могут смещаться.

В силу малого размера изображений, стилем графики был выбран пиксель-арт. Для лёгкой визуально идентификации объектов, им была присвоена цветовая кодировка, позволяющая определить тип объекта не приглядываясь к его деталям.

- **Синим** обозначены игрок, записки и бонусы. Игрок имеет вид схематичного человека и простую анимацию ходьбы.
- **Красным** обозначены вирусы. Изображение вируса представляет собой монстра с двумя щупальцами. Анимация движения вируса включает собой движение щупалец, а так же смещение половины

изображения по оси y, симулируя глюки отображения графики.

- **Оранжевым** обозначены двери и портал.
- **Залёным** обозначены стены. В игре представлено 2 вида стен. Они имеют одинаковую механику поведения и нужны для визуального разнообразия.

Интерфейс же выполнен в разных оттенках зелёного: яркие и насыщенные для текста и иконок, тёмный для фона. Этот стиль является аллюзией на старые компьютерные терминалы или, скорее, их представление в массовой культуре.

Заключение

В данном проекте нами был реализован прототип игры. Уже на данном этапе игра является законченной и работоспособной программой, выполняющей свои функции по развлечению игрока и популяризации темы компьютерной безопасности. Однако, на каждом этапе разработки мы руководствовались принципом модульности и расширяемости. Набор классов, написанных для этого проекта, позволяет с лёгкостью модифицировать существующие элементы и добавлять новые, а системы загрузки информации из сторонних файлов открывают возможность добавления неограниченного количества новых уровней без внесения каких-либо изменений в код программы.

Весь код игры, а так же все графические и аудио материалы уникальны и были созданы специально для этого проекта. Мы не использовали материалы, защищённые авторским правом, распространяемые по открытой лицензии, а также результаты работы нейронных сетей в нашем проекте.