**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

Курсовая работа

По дисциплине «Алгоритмы и структуры данных»

Тема «Компьютерный симулятор фермы на PyGame»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.22/2398-36

Листов 5

**Руководитель разработки**:

доцент каф. ИВК, к.т.н., доцент

*Шишкин Вадим Викторинович*

« » 2023 г.

**Исполнитель**:

студент гр. ИСТбд-23

*Лисин Георгий Олегович*

« » 2023 г.

**2023**

**Содержание**

Аннотация.......................................................................................1

Техническое задание.....................................................................3

Пояснительная записка ................................................................8

Руководство программиста ..........................................................13

Текст программы............................................................................25

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

Тема «Компьютерный симулятор фермы на PyGame»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.22/2398-36 ТЗ-03

Листов 5

**Исполнитель**:

студент гр. ИСТбд-23

*Лисин Г. О.*

« » 2023 г.

**2023**

**Введение**

Компьютерный симулятор фермы.

Приложение должно соответствовать требованиям симулятора фермы.

**Поле и игрок.** Играет один игрок. Игра происходит на отрисованном поле с внутриигровыми объектами. Размер карты составляет 4800 х 4224 пикселей и делится на невидимые для игрока плитки 64 на 64 пикселя. Игра ведётся в приделах заданного игрового поля, окружённого невидимыми стенами. Игрок может передвигаться, использовать инструменты и взаимодействовать с внутриигровыми объектами путём нажатия клавиш на клавиатуре.

**Растения.** Каждое растение имеет семена, который можно покупать в магазине. У каждого из таких растений есть своё уникальное время созревания, исчисляемое в игровых днях. Каждое растение имеет свои фазы роста и уникальные спрайты. У каждого растения есть своя цена продажи, как и цена покупки семян. Каждое растение имеет свои «коллизии» иначе говоря «хитбоксы» на определённой фазе созревания.

**Стартовая позиция.** Игрок располагается в своём доме в заданной точке, откуда может начать исследовать мир.

**Взаимодействие с игрой.** Игрок может рубить дерево, собирать яблоки, создавать грядки, сажать растения, поливать грядки, покупать семена, продавать растения, передвигаться с разной скоростью и спать, на данный момент это весь функционал, но он может расширяться.

### **Цель**. Игра по своей задумке имеет вид песочницы и не имеет конкретного финала, развитие может быть бесконечным.

**1. Основания для разработки**

Учебный план направления 09.03.02 «Информационные системы и технологии» и распоряжение по факультету «О закреплении тем курсовых работ (проектов) за студентами 2 курса ФИСТ направления 09.03.02 «Информационные системы и технологии» (профиль Информационные системы и технологии) по дисциплине «Алгоритмы и структуры данных» от 20.10.2023

**2. Требования к программе или программному изделию**

**2.1. Функциональное назначение**

Требуется разработать десктопное приложение, представляющее собой симулятор фермы с графическим интерфейсом в среде Windows.

**2.2 Требования к функциональным характеристикам**

2.2.1 Требования к структуре приложения

Приложение должно быть разработано в виде нескольких модулей, взаимодействующих между собой с использованием дополнительных информационных файлов.

2.2.2 Требования к составу функций приложения

В приложении должны быть реализованы в графическом режиме следующие основные функции:

- регистрация/авторизация пользователя;

- отрисовка игрового поля;

- взаимодействие с пользователем;

2.2.2 Требования к организации информационного обеспечения, входных и выходных данных

В приложении должен быть реализован графический интерфейс взаимодействия с пользователем. Отдельно выделены папки под графические файлы, шрифт, заготовку объектов и карты, аудио эффектов, а также для самого кода. Логин и пароль пользователя должны вводиться с клавиатуры. Логины и пароли зарегистрированных пользователей должны храниться в отдельном файле или базе данных в зашифрованном виде.

**2.3 Требования к надёжности**

Поддержка непрерывной и стабильной работы компьютера.

**2.4 Требования к информационной и программной совместимости**

Рекомендуется к использованию на Windows 10/11.

При создании программы используются встроенные библиотеки “random”, “os”, “sys”, “hashlib”. И сторонние библиотеки “pygame 2.5.2”, “pytmx” взятые как основа графического интерфейса.

Разработка ведётся в “PyCharm community edition 2023.3” на версии языка программирования Python 3.10.5.

**2.5. Требования к маркировке и упаковке**

Определяются заданием на курсовую работу.

**2.6 Требования к транспортированию и хранению**

2.6.1 Условия транспортирования

Требования к условиям транспортирования не предъявляются

2.6 2 Условия хранения

Диск CD-R должен храниться при комнатной температуре, в диапазоне от 20°C до 25°C. Рекомендуется хранить диск в условиях с относительной влажностью воздуха от 20% до 50%. Диск CD-R должен храниться в темном месте, защищенном от прямых солнечных лучей и других источников яркого света. Для предотвращения повреждения диска CD-R рекомендуется хранить его в специальных пластиковых коробках или футлярах, предназначенных для хранения CD-дисков.

2.6 3 Сроки хранения

Срок хранения – до июля 2024 года**.**

**3. Требования к программной документации**

Определяется заданием на курсовую работу.

**4. Стадии и этапы разработки**

Определяется заданием на курсовую работу.

**5. Порядок контроля и приёмки**

Определяется заданием на курсовую работу.

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

Курсовая работа

По дисциплине «Алгоритмы и структуры данных»

Тема «Компьютерный симулятор фермы на PyGame»

Пояснительная записка

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.22/2398-36 ПЗ-03

Листов 5

**Руководитель разработки**:

доцент каф. ИВК, к.т.н., доцент

*Шишкин Вадим Викторинович*

« » 2023 г.

**Исполнитель**:

студент гр. ИСТбд-23

*Лисин Георгий Олегович*

« » 2023 г.

**2023**

**Введение**

Приложение «Cat’s Farm» реализует функционал базового игрового симулятора фермы.

В качестве подхода для разработки была выбрана каскадная модель разработки («Водопад»). Каскадная модель была выбрана из-за своей простоты, она позволяет наглядно представить объём работы и сроки выполнения, а также эффективно разбить проект на несколько подзадач.

Приложение «Cat’s Farm» представляет собой игру для одного игрока, которая позволит ему создать свою уютную ферму и «разбогатеть».

**1. Проектная часть**

**1.1 Постановка задачи на разработку приложения**

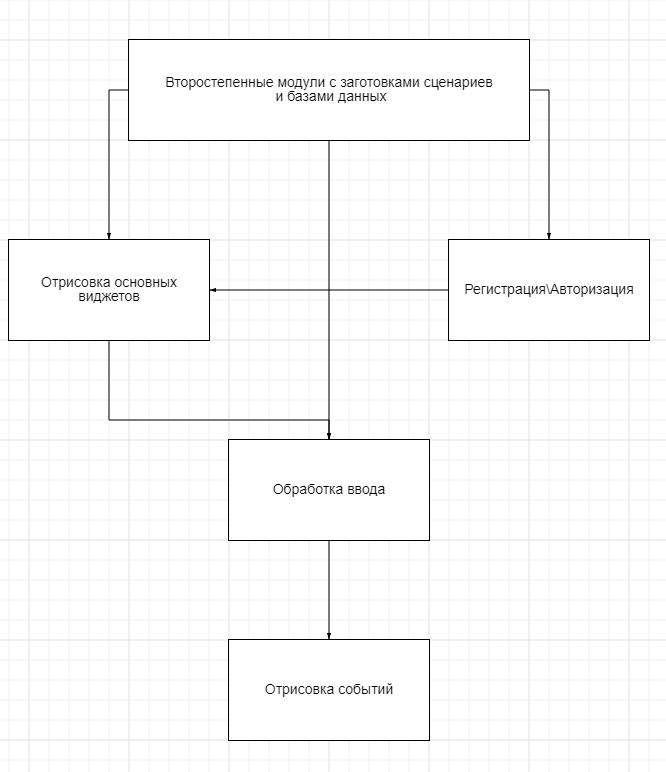
Определяется заданием на курсовую работу

**1.2 Математические методы**

В качестве математической модели для представления и отрисовки поля был выбран двумерный массив, он позволяет легко записать положение всех грядок взаимодействовать с их содержимым, путём записи информации. В массиве плиты, в которых нет грядки пустые, те, которые подлежат созданию грядки имеют ‘F’, те, которые были вскопаны ‘X’, те, которые были политы ‘W’, те, которые были засажены ‘P’.

**1.3 Архитектура и алгоритмы**

1.3.1. Архитектура



1. Регистрация/Авторизация.

Интерфейс, созданный с помощью библиотеки PyGame позволяющий пользователю зарегистрировать новый аккаунт и сыграть с помощью уже существующего.

2. Отрисовка основных виджетов.

Вывод на экран игрового поля и внутриигровых объектов.

3. Обработка ввода.

Алгоритмы проверки взаимодействия пользователя с внутриигровыми объектами и последующая реакция игры на действия игрока.

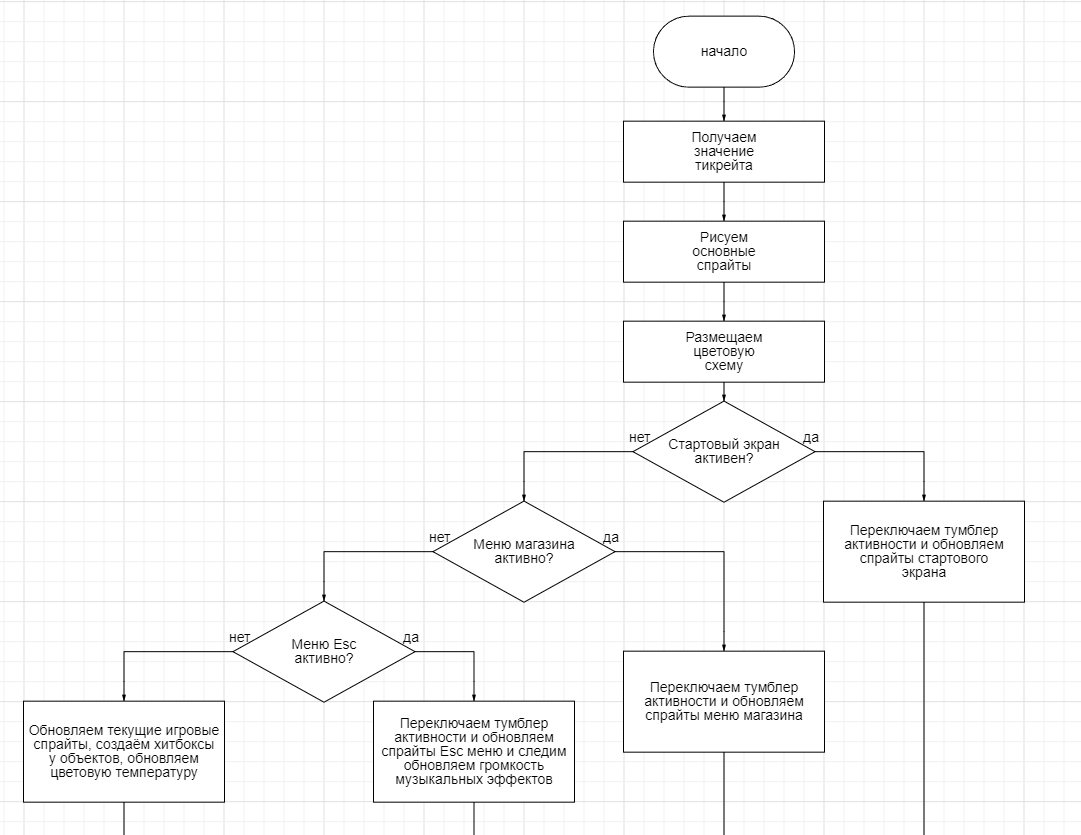
4. Отрисовка событий.

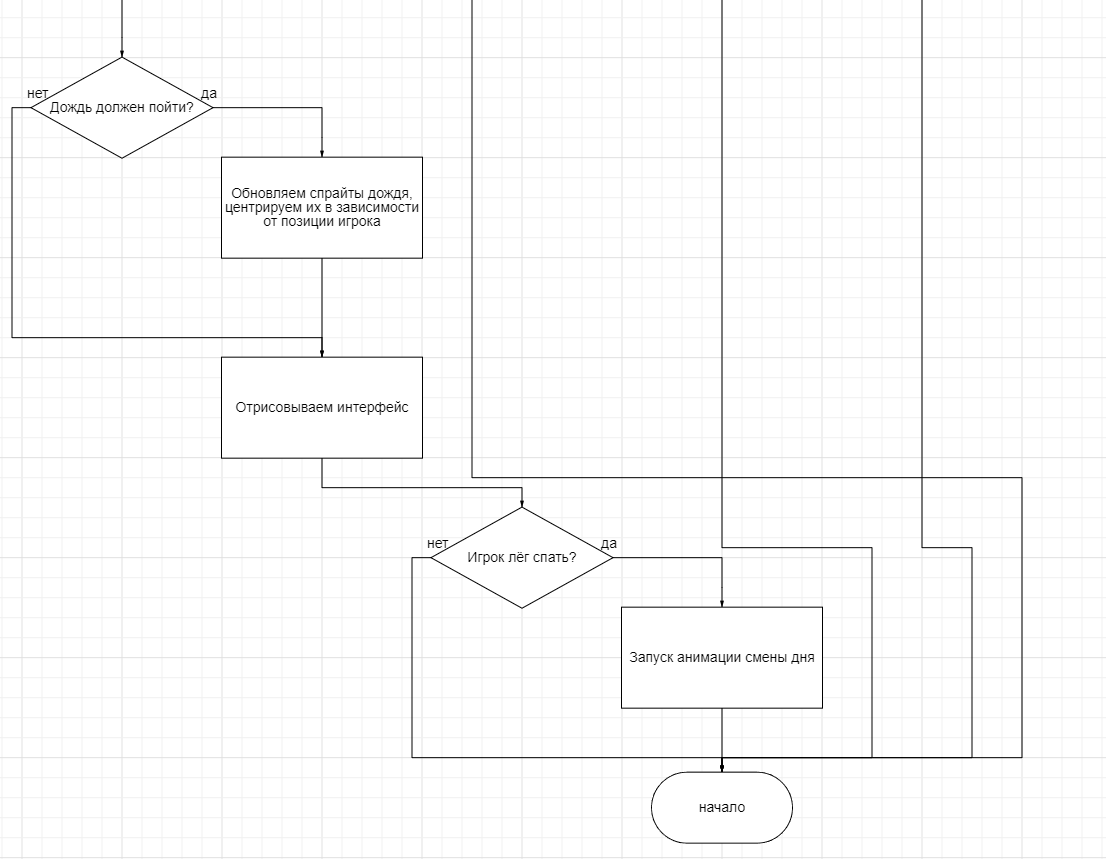
Вывод на экран результат внутриигрового взаимодействия.

1.3.2.1. Алгоритм проверки регистрации

Данный алгоритм необходим для проверки регистрации пользователя. Алгоритм принимает логин и пароль, шифрует пароль, сверяет полученные данные с данными о зарегистрированных пользователях. Если пользователь с таким логином найден и пароль соответствует записанному в файл с зарегистрированными, то алгоритм позволяет начать игру, иначе выводит предупреждение о том, что данные некорректны.

1.3.2.2. Главный алгоритм отрисовки





**1.4 Тестирование**

Весь процесс тестирования проходил вручную, без привлечения специального ПО. На протяжении всего хода разработки, использовался метод белого ящика, так как в любом время имелся доступ ко всем компонентам программы. Всё тестирование выполнялось интуитивным методом, без подготовки специальных тестов.

На протяжении всего хода разработки, по мере добавления новых функций программы, использовалось системное тестирование новых функций, для устранения возникших в ходе написания ошибок. После положительных результатов тестирования функция считалась внедренной.

**2. Источники, использованные при разработке**

Разработка логических компьютерных игр с графическим интерфейсом в среде Питон: https://pg1.readthedocs.io/en/latest/index.html

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

Курсовая работа

По дисциплине «Алгоритмы и структуры данных»

Тема «Компьютерный симулятор фермы на PyGame»

Руководство программиста

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.22/2398-36 РП-01

Листов 12

**Исполнитель**:

студент гр. ИСТбд-23

*Лисин Георгий Олегович*

« » 2023 г.

**2023**

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Компьютерный симулятор фермы.

Приложение должно соответствовать требованиям симулятора фермы.

**Поле и игрок.** Играет один игрок. Игра происходит на отрисованном поле с внутриигровыми объектами. Размер карты составляет 4800 х 4224 пикселей и делится на невидимые для игрока плитки 64 на 64 пикселя. Игра ведётся в приделах заданного игрового поля, окружённого невидимыми стенами. Игрок может передвигаться, использовать инструменты и взаимодействовать с внутриигровыми объектами путём нажатия клавиш на клавиатуре.

**Растения.** Каждое растение имеет семена, который можно покупать в магазине. У каждого из таких растений есть своё уникальное время созревания, исчисляемое в игровых днях. Каждое растение имеет свои фазы роста и уникальные спрайты. У каждого растения есть своя цена продажи, как и цена покупки семян. Каждое растение имеет свои «коллизии» иначе говоря «хитбоксы» на определённой фазе созревания.

**Стартовая позиция.** Игрок располагается в своём доме в заданной точке, откуда может начать исследовать мир.

**Взаимодействие с игрой.** Игрок может рубить дерево, собирать яблоки, создавать грядки, сажать растения, поливать грядки, покупать семена, продавать растения, передвигаться с разной скоростью и спать, на данный момент это весь функционал, но он может расширяться.

### **Цель**. Игра по своей задумке имеет вид песочницы и не имеет конкретного финала, развитие может быть бесконечным.

**1.2 Условия, необходимые для использования приложения**

Рекомендуется к использованию на Windows 10/11.

При создании программы используются встроенные библиотеки “random”, “os”, “sys”, “hashlib”. И сторонние библиотеки “pygame 2.5.2”, “pytmx” взятые как основа графического интерфейса.

Разработка ведётся в “PyCharm community edition 2023.3” на версии языка программирования Python 3.10.5.

**2. Характеристики программы**

**2.1 Характеристики приложения**

Значимых строк кода: 2000

Структуры данных – массивы и списки.

Использованные библиотеки:

“os” – библиотека функций для работы с операционной системой. Методы, включенные в неё, позволяют определять тип операционной системы, получать доступ к переменным окружения, управлять директориями и файлами.

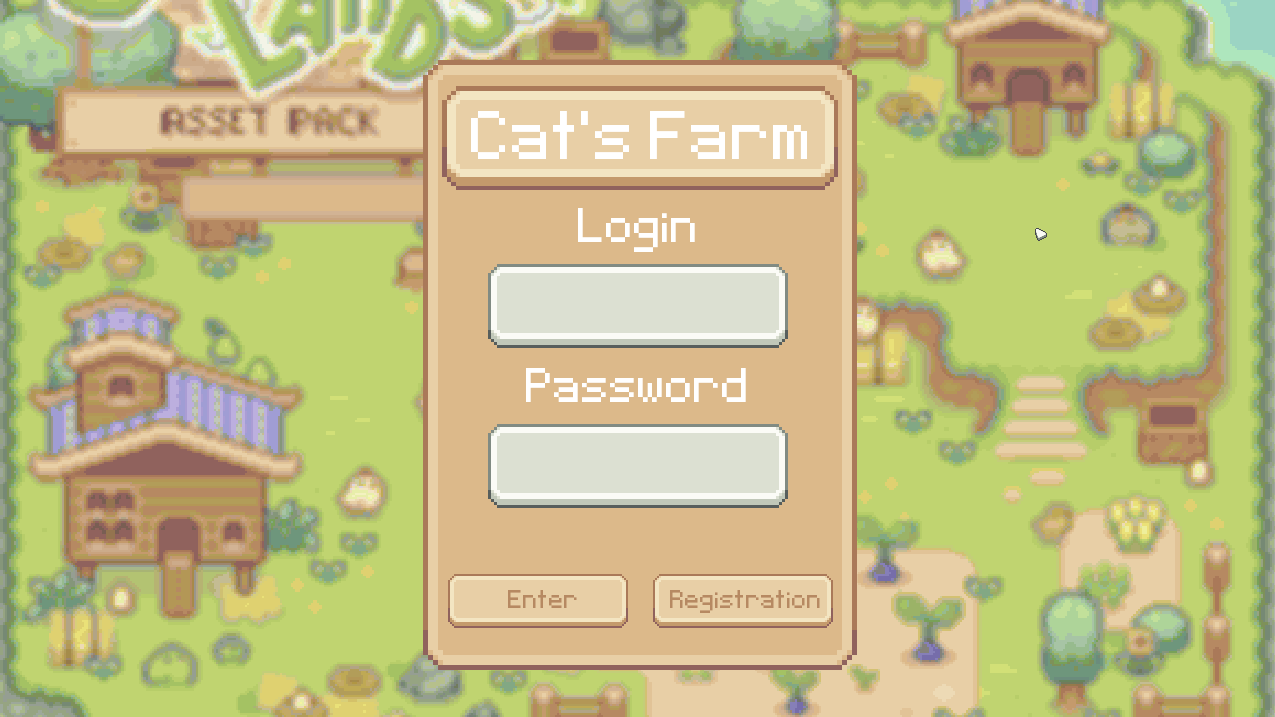
“pygame 2.5.2” - библиотека для разработки графического интерфейса на языке Python. Методы, включенные в неё, позволяют создавать окна, размещать на них виджеты, настраивать параметры окна и виджетов.

“hashlib” – библиотека для кодирования пользовательских данных.

“pytmx” – библиотека для работы с графическим редактором Tiled и импортом данных из него.

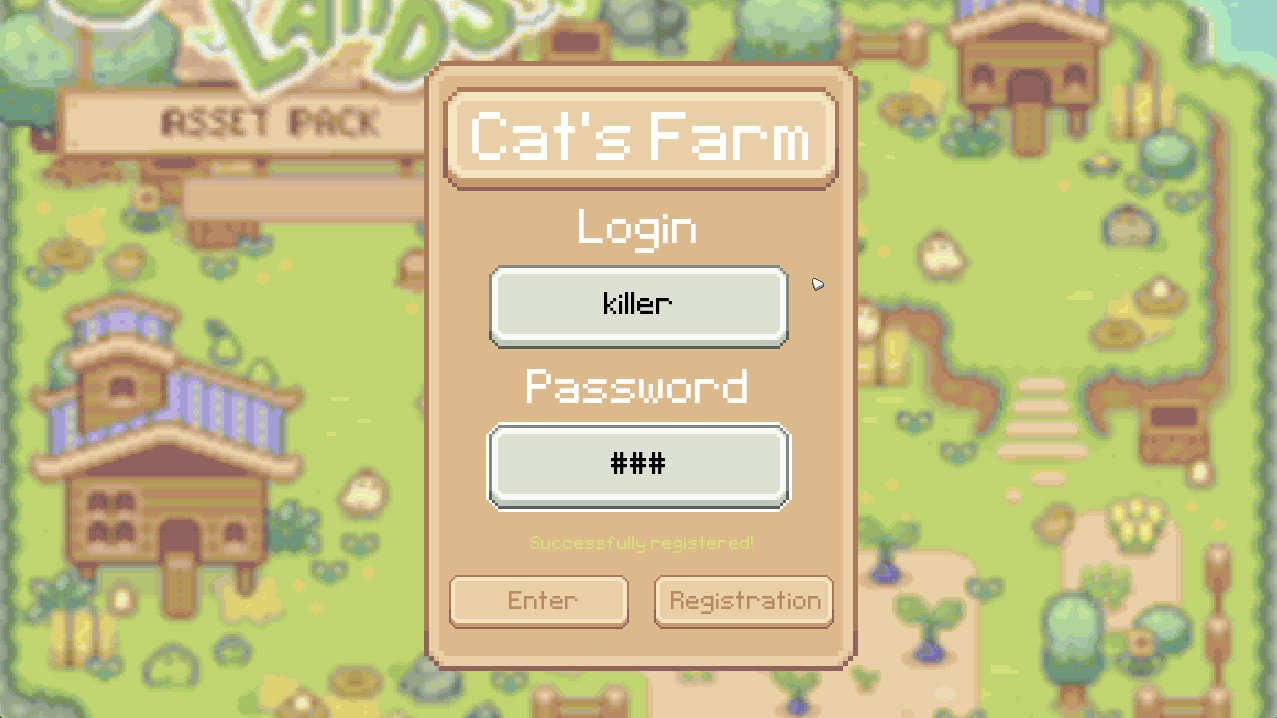
“sys” – позволяет настраивать и контролировать взаимодействие программы с операционной системы.

Работа приложения начинается с окна авторизации и регистрации

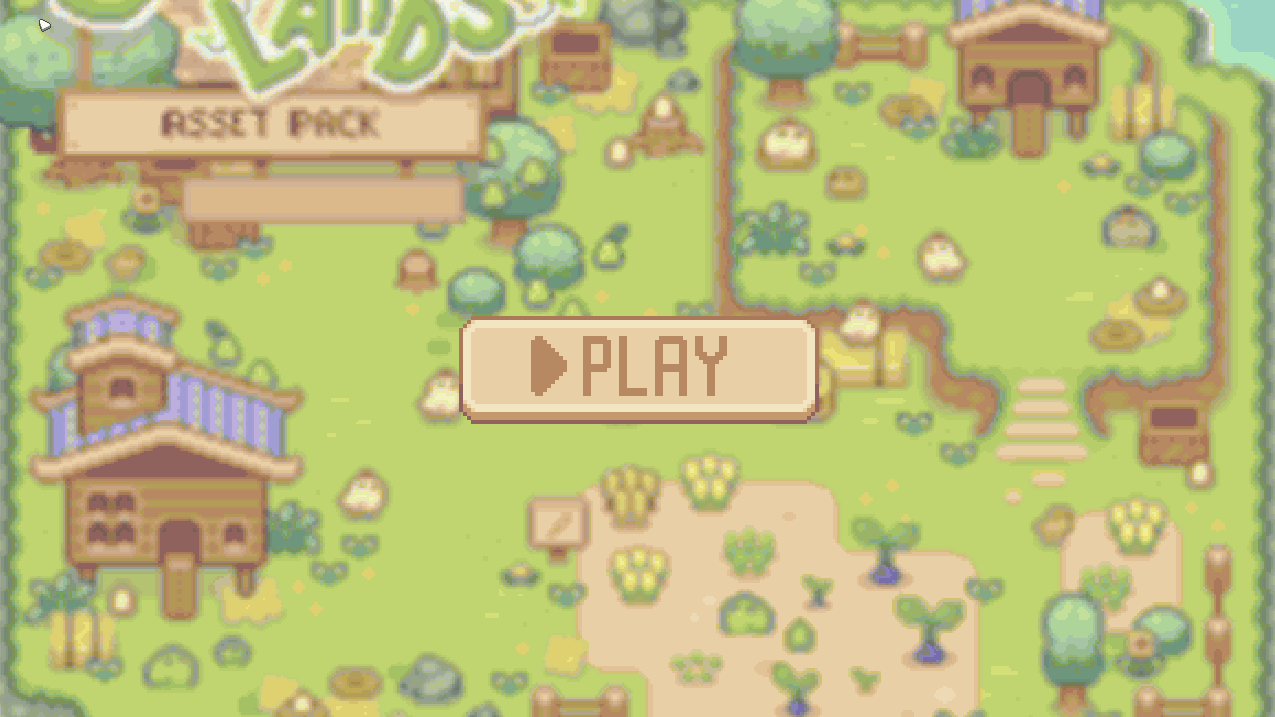


Успешная регистрация

Успешная регистрация



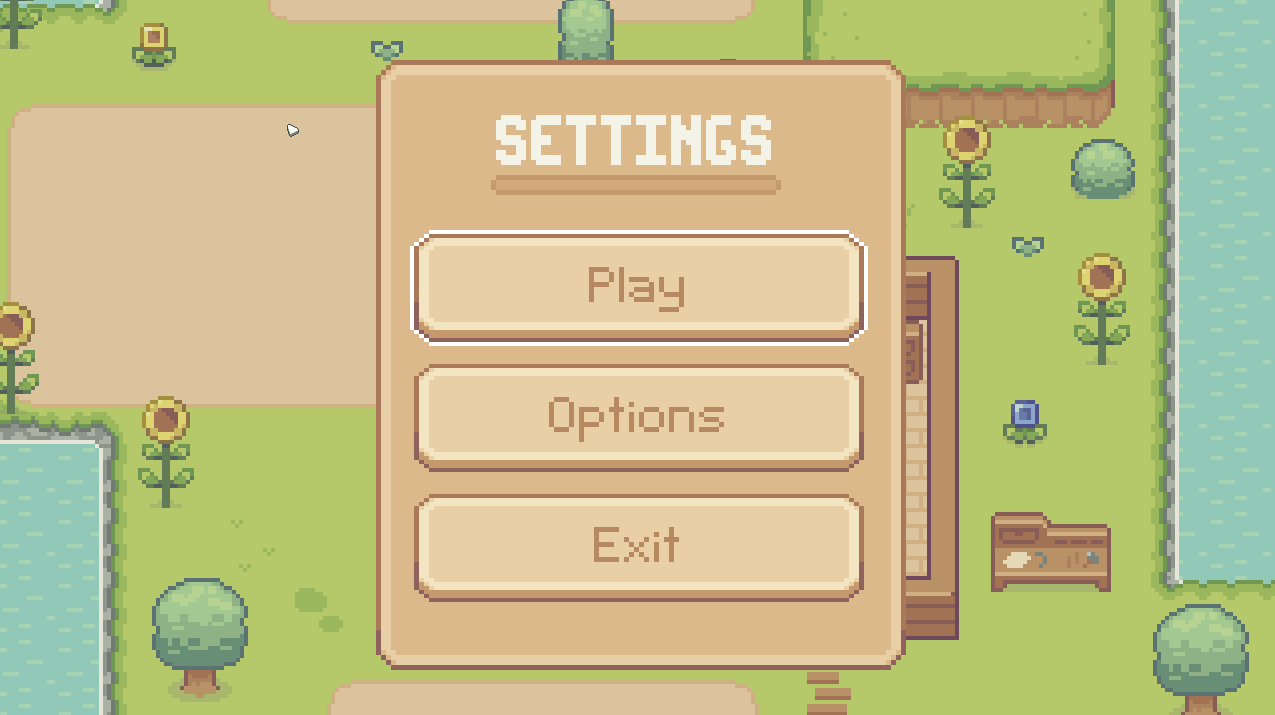
Успешная авторизация



Отрисовка игры



Настройки, открывающиеся на кнопку Esc



Использование инструментов на пробел



Создание грядки на пробел



Поливка грядки на пробел



Смена инструментов происходит на клавиши q и e

Посадка растений на tab

Взаимодействие с кроватью и торговцем f

**2.2 Особенности реализации приложения**

В программе используется структура данных массив, эта структура была выбрана ввиду удобной навигации для отрисовки и проверок наличия объектов в грядке, т.к. двумерном массиве есть возможность поиска по строке и столбцу.

**3. Обращение к программе**

main.run – запуск игры

level.setup – создание всех внутриигровых объектов через доп. методы.

level.player­\_add – добавляет подаваемое на вход значение в инвентарь персонажа

level.toggle\_esc\_menu – переключатель режима работы меню Esc

level.toggle\_shop – переключатель режима работы меню магазина

level.toggle\_start\_menu – переключатель режима работа стартового меню

level.reset – функция смены дня, обновляющая деревья, сбрасывая время дня и уменьшающая рост растений.

level.plant\_collision – функция, которая следит за тем, столкнулся ли игрок с выросшими растениями, если столкнулся, то удаляет их и добавляет игроку единицу урожая через level.player\_add

level.run – основная функция отрисовки игровых событий, ключевая обобщающая функция

level.custom\_draw – функция, которая заменяет встроенную функцию отрисовки из pygame для сортировки объектов по их нахождению на оси y

player.use\_tool – функция которая задействует сценарий использования инструмента

player.get\_target\_pos – функция, которая в зависимости от выбранного инструмента создаёт невидимую точку «удара» инструмента для взаимодействия с грядками и деревьями

player.use\_seed – функция вызывающая сценарий посадки растения

player.import\_assets – функция импорта набора картинок для анимации персонажа.

player.animate – функция анимации действий персонажа

player.input – функция считывания ввода с клавиатуры и основного взаимодействия игры с игроком

player.get\_status – функция для корректного отображения анимации в разных сценариях

player.update\_timer – функция обновления функция класса Timer

player.collision – функция предотвращения прохождения персонажа через объекты

player.move – функция осуществляющая перемещения персонажа по карте

player.update – основная функция класса, которая обновляет почти все вышеперечисленные функции класса

start\_menu.draw\_screen – функция отрисовки элементов стартового экрана

start\_menu.input – функция считывая нажатий на клавиатуру и передвижения мыши, её нажатия

start\_menu.check\_account – функция проверки наличия введённых данных в существующих аккаунтах

start\_menu.registrate – функция регистрации нового аккаунта пользователя

start\_menu.update – функция обновления основных функций

support.import\_folder – функция удобного импорта данных в массив

support.import\_folder\_dict – функция удобного импорта данных в список

timer.activate – функция запуска таймера

timer.deactivate – функция остановки таймера

timer.update – функция обновления накопления счётчика таймера

transition.play – функция смены цветовой температуры со временем

overlay.display – функция отрисовки основных элементов интерфейса

shop.display\_money – функция отрисовки значения денежных средств

shop.setup – функция для заготовки основных поверхностей для отрисовки

shop.input – функция считывания нажатых клавиш

shop.show\_entry - функция отрисовки основных элементов магазина

shop.update – функция обновления спрайтов магазина

soil.grow – функция роста растений (смены спрайтов)

soil.create\_soil\_grid – функция проверки и создания группы спрайтов грядок

soil.create\_hit\_rects – функция создания невидимых хитбоксов грядки

soil.get\_hit – функция создания и размещения спрайта взрыхлённой грядки

soil.water – функция создания и размещения спрайта политой грядки

soil.water\_all – функция создания и размещения спрайтов воды для всех грядок (используется, когда идёт дождь)

soil.remove\_water – функция удаления спрайтов воды

soil.remove\_soil – функция удаления спрайтов растений

soil.check\_watered – функция проверки наличия воды в грядке

soil.plant\_seed – функция создания и размещения спрайтов растения

soil.update\_plants – функция вызова функции обновления роста для всех растений

soil.create\_soil\_tiles – первоначальная функция создания и размещения спрайтов грядки

sprites.update – функция обновления спрайтов частиц

sprites.damage – функция получения деревом урона и удаления яблока

sprites.check\_death – функция проверки наличия здоровья у дерева, в противном случае уничтожение его спрайта с заменой на спрайт пня и добавление в инвентарь игрока древесины

sprites.create\_fruit – функция создания и размещения спрайтов яблока на деревьях

sprites.update – функция обновления функций в реальном времени

esc.menu\_setup – функция заготовки текстовых объектов для отрисовки интерфейса меню

esc.menu\_draw\_menu – функция отрисовки основного интерфейса меню в зависимости от открытой вкладки

esc.input – функция считывания нажатий игрока

esc.update – функция обновления функций в реальном времени

Библиотеки, используемые во время разработки приложения:

“os” – библиотека функций для работы с операционной системой. Методы, включенные в неё, позволяют определять тип операционной системы, получать доступ к переменным окружения, управлять директориями и файлами.

“pygame 2.5.2” - библиотека для разработки графического интерфейса на языке Python. Методы, включенные в неё, позволяют создавать окна, размещать на них виджеты, настраивать параметры окна и виджетов.

“hashlib” – библиотека для кодирования пользовательских данных.

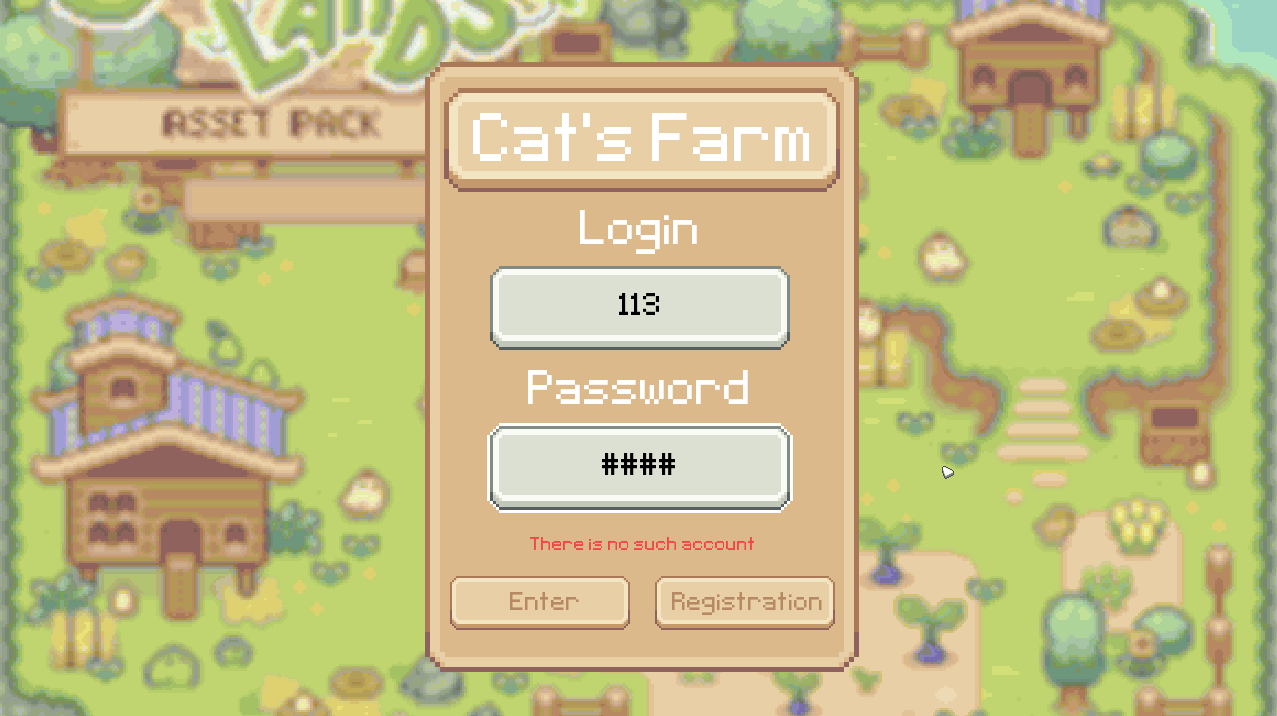
“pytmx” – библиотека для работы с графическим редактором Tiled и импортом данных из него.

“sys” – позволяет настраивать и контролировать взаимодействие программы с операционной системы.

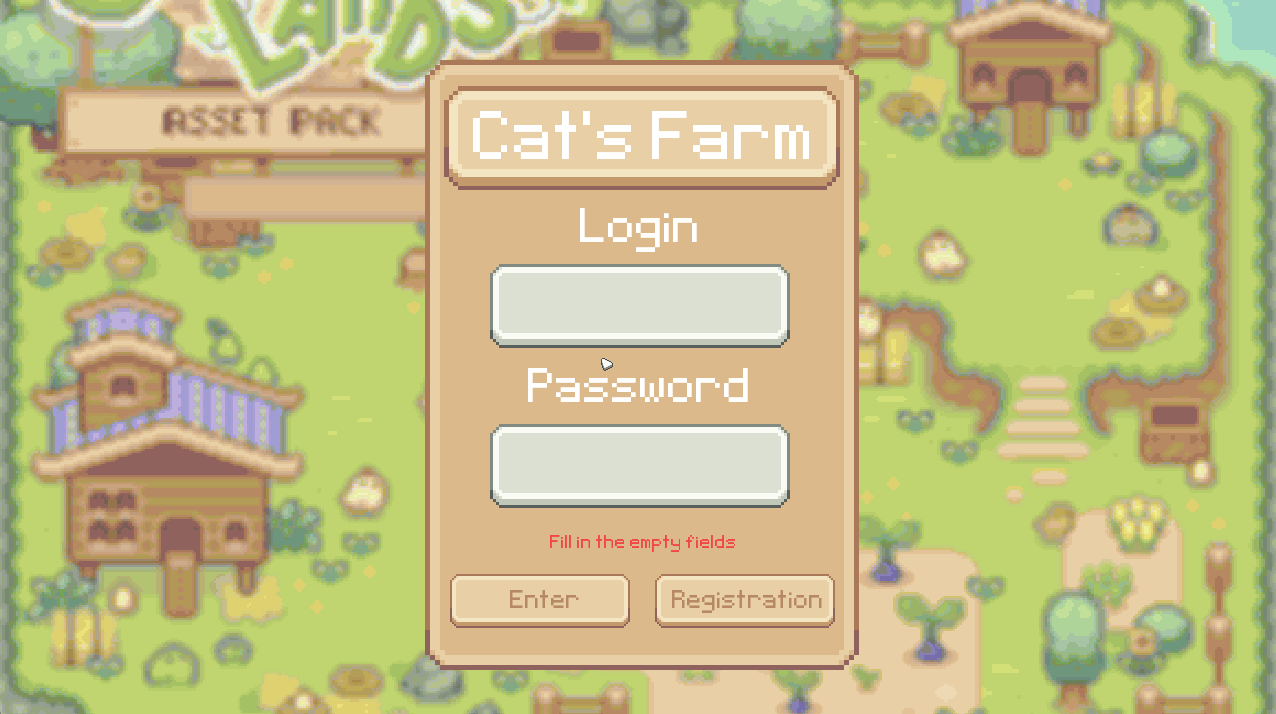
Основные виджеты библиотеки pygame: image, rect, render.

**4. Сообщения**

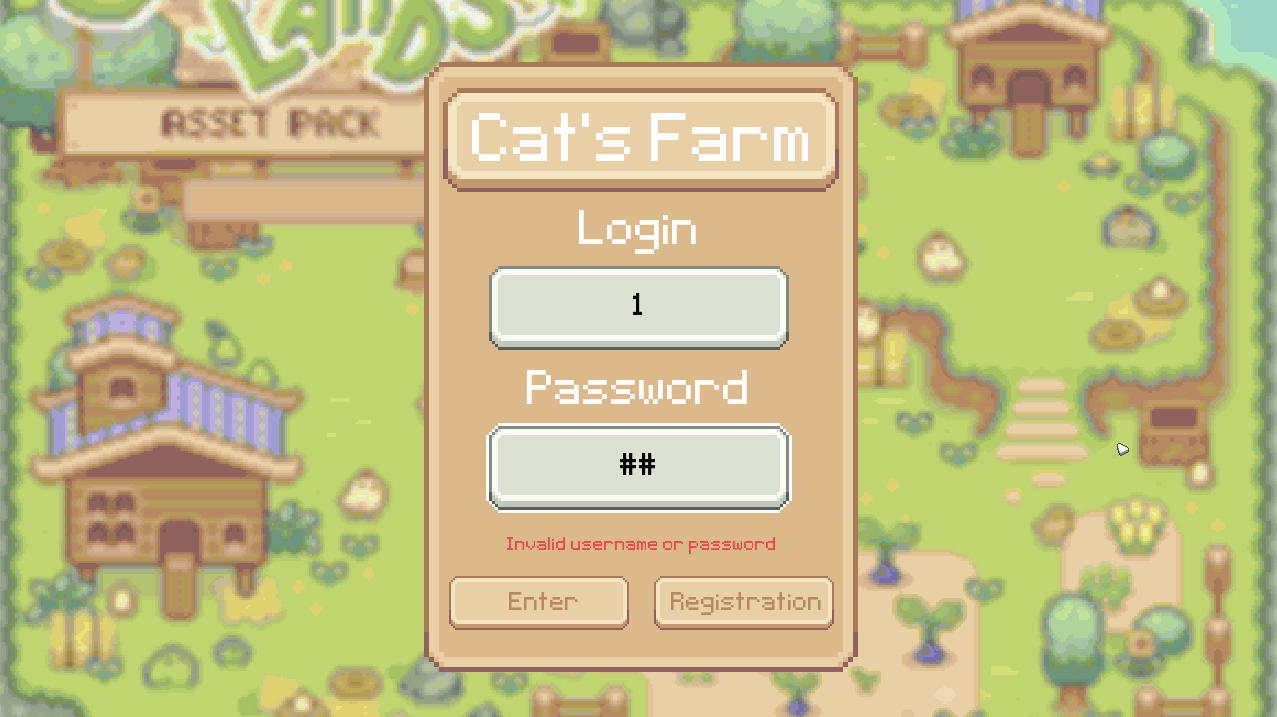
Пользователь не найден:

****

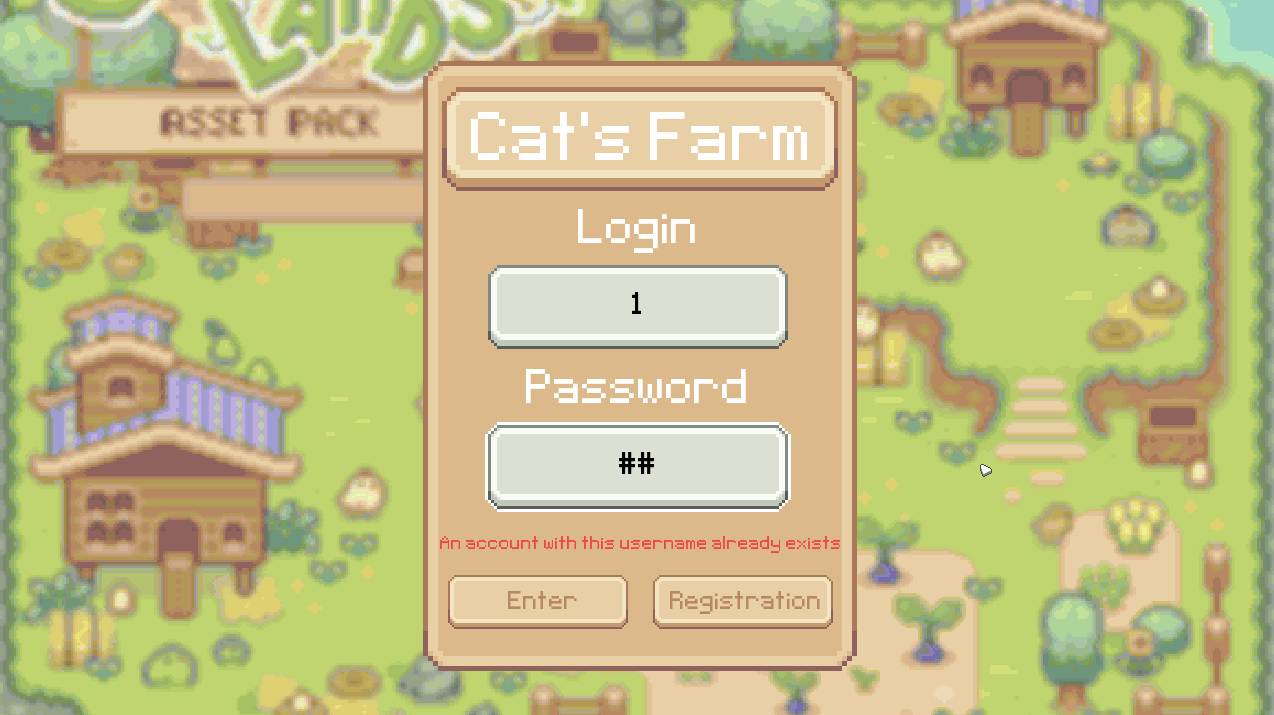
Поле для ввода пароля или логина пустое:



Введен неверный пароль:



Аккаунт с таким именем пользователя уже существует:



**Текст программы**

**main.py**

*import* sys  
*import* pygame  
*from* settings *import* \*  
*from* level *import* Level  
  
  
*class* Game:  
 *def \_\_init\_\_*(*self*):  
 pygame.init()  
 *self*.screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))pygame.display.set\_caption('Ферма Кэта')  
 *self*.clock = pygame.time.Clock()  
 *self*.level = Level()  
  
 *def* run(*self*):  
 *while True*:  
 *for* event *in* pygame.event.get():  
 *if* event.type == pygame.QUIT:  
 pygame.quit()  
 sys.exit()  
  
 dt = *self*.clock.tick() / 1000  
 *self*.level.run(dt)  
 pygame.display.update()  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 game = Game()  
 game.run()

**level.py**

*from* shop *import* Shop  
*from* support *import* \*  
*from* settings *import* \*  
*from* sky *import* Rain, Sky  
*from* player *import* Player  
*from* random *import* randint  
*from* soil *import* SoilLayer  
*from* overlay *import* Overlay  
*from* esc\_menu *import* EscMenu  
*from* start\_menu *import* StartMenu  
*from* transition *import* Transition  
*from* pytmx.util\_pygame *import* load\_pygame  
*from* sprites *import* Generic, WildFlower, Tree, Interaction, Particle, Collision  
  
  
*class* Level:  
 *def \_\_init\_\_*(*self*):  
 *# get the display surface  
 self*.display\_surface = pygame.display.get\_surface()  
  
 *# sprite groups  
 self*.all\_sprites = CameraGroup()  
 *self*.collision\_sprites = pygame.sprite.Group()  
 *self*.tree\_sprites = pygame.sprite.Group()  
 *self*.interaction\_sprites = pygame.sprite.Group()  
  
 *self*.soil\_layer = SoilLayer(*self*.all\_sprites, *self*.collision\_sprites)  
 *self*.setup()  
 *self*.overlay = Overlay(*self*.player)  
 *self*.transition = Transition(*self*.reset, *self*.player)  
  
 *# sky  
 self*.rain = Rain(*self*.all\_sprites)  
 *self*.raining = randint(0, 10) > 10  
 *self*.soil\_layer.raining = *self*.raining  
 *self*.sky = Sky()  
  
 *# start menu  
 self*.start\_menu = StartMenu(*self*.toggle\_start\_menu)  
 *self*.start\_menu\_active = *True  
  
 # esc menu  
 self*.esc\_menu = EscMenu(*self*.player, *self*.toggle\_esc\_menu)  
 *self*.esc\_menu\_active = *False  
  
 # shop  
 self*.shop\_menu = Shop(*self*.player, *self*.toggle\_shop)  
 *self*.shop\_active = *False  
  
 # import  
 self*.cursor\_surf = pygame.image.load(f'../graphics/overlay/cursor.png').convert\_alpha()  
  
 *# music  
 self*.rain\_sound = pygame.mixer.Sound('../audio/rain.mp3')  
 *self*.rain\_sound.set\_volume(SOUND\_VOLUME['Rain'])  
 *self*.success = pygame.mixer.Sound('../audio/success3.mp3')  
 *self*.success.set\_volume(SOUND\_VOLUME['Success'])  
 *self*.music = pygame.mixer.Sound('../audio/bg\_music2.mp3')  
 *self*.music.set\_volume(SOUND\_VOLUME['Music'])  
 *self*.music.play(loops=-1)  
  
 *def* setup(*self*):  
 *# load map tmx* tmx\_data = load\_pygame('../data/map.tmx')  
  
 *# house  
 for* layer *in* ['HouseFloor', 'HouseFurnitureBottom']:  
 *for* x, y, surf *in* tmx\_data.get\_layer\_by\_name(layer).tiles():  
 Generic((x \* TILE\_SIZE, y \* TILE\_SIZE), surf, *self*.all\_sprites, LAYERS['house bottom'])  
  
 *for* layer *in* ['HouseWalls', 'HouseFurnitureTop']:  
 *for* x, y, surf *in* tmx\_data.get\_layer\_by\_name(layer).tiles():  
 Generic((x \* TILE\_SIZE, y \* TILE\_SIZE), surf, *self*.all\_sprites)  
  
 *# Fence  
 for* x, y, surf *in* tmx\_data.get\_layer\_by\_name('Fence').tiles():  
 Generic((x \* TILE\_SIZE, y \* TILE\_SIZE), surf, [*self*.all\_sprites, *self*.collision\_sprites])  
  
 *# Trees  
 for* obj *in* tmx\_data.get\_layer\_by\_name('Trees'):  
 Tree(pos=(obj.x, obj.y),  
 surf=obj.image,  
 groups=[*self*.all\_sprites, *self*.collision\_sprites, *self*.tree\_sprites],  
 name=obj.name,  
 player\_add=*self*.player\_add)  
  
 *# Wild Flowers  
 for* obj *in* tmx\_data.get\_layer\_by\_name('Decoration'):  
 WildFlower((obj.x, obj.y), obj.image, [*self*.all\_sprites, *self*.collision\_sprites])  
  
 *# Collision tiles  
 for* x, y, surf *in* tmx\_data.get\_layer\_by\_name('Collision').tiles():  
 Collision('Collision', (x \* TILE\_SIZE, y \* TILE\_SIZE), pygame.Surface((TILE\_SIZE, TILE\_SIZE)),  
 *self*.collision\_sprites)  
 *for* x, y, surf *in* tmx\_data.get\_layer\_by\_name('Collision up').tiles():  
 Collision('Collision up', (x \* TILE\_SIZE, y \* TILE\_SIZE), pygame.Surface((TILE\_SIZE, TILE\_SIZE)),  
 *self*.collision\_sprites)  
 *for* x, y, surf *in* tmx\_data.get\_layer\_by\_name('Collision down').tiles():  
 Collision('Collision down', (x \* TILE\_SIZE, y \* TILE\_SIZE), pygame.Surface((TILE\_SIZE, TILE\_SIZE)),  
 *self*.collision\_sprites)  
 *for* x, y, surf *in* tmx\_data.get\_layer\_by\_name('Collision right').tiles():  
 Collision('Collision right', (x \* TILE\_SIZE, y \* TILE\_SIZE), pygame.Surface((TILE\_SIZE, TILE\_SIZE)),  
 *self*.collision\_sprites)  
 *for* x, y, surf *in* tmx\_data.get\_layer\_by\_name('Collision left').tiles():  
 Collision('Collision left', (x \* TILE\_SIZE, y \* TILE\_SIZE), pygame.Surface((TILE\_SIZE, TILE\_SIZE)),  
 *self*.collision\_sprites)  
 *for* x, y, surf *in* tmx\_data.get\_layer\_by\_name('Collision corner').tiles():  
 Collision('Collision corner', (x \* TILE\_SIZE, y \* TILE\_SIZE), pygame.Surface((TILE\_SIZE, TILE\_SIZE)),  
 *self*.collision\_sprites)  
  
 *# Player  
 for* obj *in* tmx\_data.get\_layer\_by\_name('Player'):  
 *if* obj.name == 'Start':  
 *self*.player = Player(  
 pos=(obj.x, obj.y),  
 group=*self*.all\_sprites,  
 collision\_sprites=*self*.collision\_sprites,  
 tree\_sprites=*self*.tree\_sprites,  
 interaction=*self*.interaction\_sprites,  
 soil\_layer=*self*.soil\_layer,  
 toggle\_shop=*self*.toggle\_shop,  
 start\_menu=*self*.toggle\_esc\_menu)  
  
 *if* obj.name == 'Bed':  
 Interaction((obj.x, obj.y), (obj.width, obj.height), *self*.interaction\_sprites, obj.name)  
  
 *if* obj.name == 'Trader':  
 Interaction((obj.x, obj.y), (obj.width, obj.height), *self*.interaction\_sprites, obj.name)  
  
 *# Ground* Generic(  
 pos=(0, 0),  
 surf=pygame.image.load('../graphics/world/ground.png').convert\_alpha(),  
 groups=*self*.all\_sprites,  
 z=LAYERS['ground']  
 )  
  
 *def* player\_add(*self*, item, cnt=1):  
 *self*.player.item\_inventory[item] += cnt  
 *self*.success.play()  
  
 *def* toggle\_esc\_menu(*self*):  
 *self*.esc\_menu\_active = *not self*.esc\_menu\_active  
  
 *def* toggle\_shop(*self*):  
 *self*.shop\_active = *not self*.shop\_active  
  
 *def* toggle\_start\_menu(*self*):  
 *self*.start\_menu\_active = *not self*.start\_menu\_active  
  
 *def* reset(*self*):  
 *# plants  
 self*.soil\_layer.update\_plants()  
  
 *# soil  
 self*.soil\_layer.remove\_soil()  
 *self*.soil\_layer.remove\_water()  
 *self*.raining = randint(0, 10) > 7  
 *self*.soil\_layer.raining = *self*.raining  
 *if self*.raining:  
 *self*.rain\_sound.play(loops=-1)  
 *self*.soil\_layer.water\_all()  
 *else*:  
 *self*.rain\_sound.play(loops=-1)  
 *self*.rain\_sound.stop()  
  
 *# appel on the trees  
 for* tree *in self*.tree\_sprites.sprites():  
 *for* apple *in* tree.apple\_sprites.sprites():  
 apple.kill()  
 tree.create\_fruit()  
  
 *# sky  
 self*.sky.start\_color = [255, 255, 255]  
  
 *def* plant\_collision(*self*):  
 *if self*.soil\_layer.plant\_sprites:  
 *for* plant *in self*.soil\_layer.plant\_sprites.sprites():  
 *if* plant.harvestable *and* plant.rect.colliderect(*self*.player.hitbox):  
 *self*.player\_add(plant.plant\_type)  
 plant.kill()  
 Particle(pos=plant.rect.topleft,  
 surf=plant.image,  
 groups=*self*.all\_sprites,  
 z=LAYERS['main'])  
  
 *self*.soil\_layer.grid[plant.rect.centery // TILE\_SIZE][plant.rect.centerx // TILE\_SIZE].remove('P')  
  
 *def* run(*self*, dt):  
 *# drawing logic  
 self*.display\_surface.fill('black')  
 *self*.all\_sprites.custom\_draw(*self*.player)  
 *self*.sky.display()  
  
 *# updates  
 if self*.start\_menu\_active:  
 *self*.start\_menu.update()  
  
 *elif self*.shop\_active:  
 *self*.shop\_menu.update()  
  
 *elif self*.esc\_menu\_active:  
 *self*.esc\_menu.update()  
  
 *self*.success.set\_volume(SOUND\_VOLUME['Success'])  
 *self*.music.set\_volume(SOUND\_VOLUME['Music'])  
 *self*.soil\_layer.hoe\_sound.set\_volume(SOUND\_VOLUME['Hoe'])  
 *self*.soil\_layer.plant\_sound.set\_volume(SOUND\_VOLUME['Plant'])  
  
 *else*:  
 *# sprites  
 self*.all\_sprites.update(dt)  
 *self*.plant\_collision()  
  
 *# weather  
 self*.sky.update(dt)  
  
 *if self*.raining:  
 *self*.rain.update() *# rain  
 self*.rain.player\_pos = [*self*.player.rect.centerx, *self*.player.rect.centery]  
  
 *# overlay  
 self*.overlay.display()  
  
 *# transition overlay  
 if self*.player.sleep:  
 *self*.transition.play()  
  
 *# mouse* pygame.mouse.set\_visible(*False*)  
 *self*.display\_surface.blit(*self*.cursor\_surf, (pygame.mouse.get\_pos()))  
  
  
*class* CameraGroup(pygame.sprite.Group):  
 *def \_\_init\_\_*(*self*):  
 super().*\_\_init\_\_*()  
 *self*.display\_surface = pygame.display.get\_surface()  
 *self*.offset = pygame.math.Vector2()  
  
 *def* custom\_draw(*self*, player):  
 *self*.offset.x = player.rect.centerx - SCREEN\_WIDTH / 2  
 *self*.offset.y = player.rect.centery - SCREEN\_HEIGHT / 2  
  
 *for* layer *in* LAYERS.values():  
 *for* sprite *in* sorted(*self*.sprites(), key=*lambda* sprite: sprite.rect.centery):  
 *if* sprite.z == layer:  
 offset\_rect = sprite.rect.copy()  
 offset\_rect.center -= *self*.offset  
 *self*.display\_surface.blit(sprite.image, offset\_rect)

**player.py**

*from* support *import* \*  
*from* settings *import* \*  
*from* timer *import* Timer  
  
  
*class* Player(pygame.sprite.Sprite):  
 *def \_\_init\_\_*(*self*, pos, group, collision\_sprites, tree\_sprites, interaction, soil\_layer, toggle\_shop, start\_menu):  
 super().\_\_init\_\_(group)  
  
 *self*.import\_assets()  
 *self*.status = 'down\_idle'  
 *self*.frame\_index = 0  
  
 *# general setup  
 self*.image = *self*.animations[*self*.status][*self*.frame\_index]  
 *self*.rect = *self*.image.get\_rect(center=pos)  
 *self*.z = LAYERS['main']  
  
 *# movement attributes  
 self*.direction = pygame.math.Vector2()  
 *self*.pos = pygame.math.Vector2(*self*.rect.center)  
 *self*.speed = 200  
  
 *# collision  
 self*.hitbox = *self*.rect.copy().inflate((-126, -70))  
 *self*.collision\_sprite = collision\_sprites  
  
 *# timers  
 self*.timers = {  
 'water use': Timer(950, *self*.use\_tool),  
 'axe use': Timer(700, *self*.use\_tool),  
 'hoe use': Timer(700, *self*.use\_tool),  
 'tool switch': Timer(200),  
 'seed use': Timer(300, *self*.use\_seed),  
 'seed switch': Timer(200)  
 }  
  
 *# tools  
 self*.tools = ['hoe', 'axe', 'water']  
 *self*.tool\_index = 1  
 *self*.selected\_tool = *self*.tools[*self*.tool\_index]  
  
 *# seeds  
 self*.seeds = ['corn', 'tomato', 'cabbage', 'carrot', 'pumpkin', 'turnip', 'zucchini', 'cucumber']  
 *self*.seed\_index = 0  
 *self*.seed\_select\_index = 0  
 *self*.selected\_seed = *self*.seeds[*self*.seed\_index]  
  
 *# inventory  
 self*.item\_inventory = {  
 'wood': 0,  
 'apple': 0,  
 'corn': 0,  
 'tomato': 0,  
 'cabbage': 0,  
 'carrot': 0,  
 'pumpkin': 0,  
 'turnip': 0,  
 'zucchini': 0,  
 'cucumber': 0  
 }  
 *self*.seed\_inventory = {  
 'corn': 5,  
 'tomato': 5,  
 'cabbage': 5,  
 'carrot': 5,  
 'pumpkin': 5,  
 'turnip': 5,  
 'zucchini': 5,  
 'cucumber': 5  
 }  
  
 *self*.money = 200  
  
 *# interaction  
 self*.tree\_sprites = tree\_sprites  
 *self*.interaction = interaction  
 *self*.sleep = *False  
 self*.soil\_layer = soil\_layer  
 *self*.toggle\_shop = toggle\_shop  
 *self*.start\_menu = start\_menu  
  
 *# sound  
 self*.switch = pygame.mixer.Sound('../audio/switch\_tool1.mp3')  
 *self*.switch.set\_volume(SOUND\_VOLUME['Switch tool'])  
 *self*.watering = pygame.mixer.Sound('../audio/water.mp3')  
 *self*.watering.set\_volume(SOUND\_VOLUME['Water'])  
 *self*.wave = pygame.mixer.Sound('../audio/Wave.mp3')  
 *self*.wave.set\_volume(SOUND\_VOLUME['Wave'])  
 *self*.choice = pygame.mixer.Sound('../audio/switch.mp3')  
 *self*.choice.set\_volume(0.05)  
  
 *def* use\_tool(*self*):  
 *if self*.selected\_tool == 'hoe':  
 *self*.soil\_layer.get\_hit(*self*.target\_pos)  
  
 *if self*.selected\_tool == 'axe':  
 *for* tree *in self*.tree\_sprites.sprites():  
 *if* tree.rect.collidepoint(*self*.target\_pos) *and* tree.alive:  
 tree.damage()  
  
 *if self*.selected\_tool == 'water':  
 *self*.soil\_layer.water(*self*.target\_pos)  
 *self*.watering.play()  
  
 *def* get\_target\_pos(*self*):  
 *if self*.selected\_tool == 'water':  
 *self*.target\_pos = *self*.rect.center + PLAYER\_WATER\_OFFSET[*self*.status.split('\_')[0]]  
 *elif self*.selected\_tool == 'hoe':  
 *self*.target\_pos = *self*.rect.center + PLAYER\_HOE\_OFFSET[*self*.status.split('\_')[0]]  
 *elif self*.selected\_tool == 'axe':  
 *self*.target\_pos = *self*.rect.center + PLAYER\_AXE\_OFFSET[*self*.status.split('\_')[0]]  
  
 *def* use\_seed(*self*):  
 target\_pos = *self*.rect.center + PLAYER\_SEED\_OFFSET[*self*.status.split('\_')[0]]  
 *if self*.seed\_inventory[*self*.selected\_seed] > 0:  
 *self*.soil\_layer.plant\_seed(target\_pos, *self*.selected\_seed, *self*.seed\_inventory, *self*.selected\_seed)  
  
 *def* import\_assets(*self*):  
 *self*.animations = {'up': [], 'down': [], 'left': [], 'right': [],  
 'right\_idle': [], 'left\_idle': [], 'up\_idle': [], 'down\_idle': [],  
 'right\_run': [], 'left\_run': [], 'up\_run': [], 'down\_run': [],  
 'right\_hoe': [], 'left\_hoe': [], 'up\_hoe': [], 'down\_hoe': [],  
 'right\_axe': [], 'left\_axe': [], 'up\_axe': [], 'down\_axe': [],  
 'right\_water': [], 'left\_water': [], 'up\_water': [], 'down\_water': []}  
  
 *for* animation *in self*.animations.keys():  
 full\_path = '../graphics/character/' + animation  
 *self*.animations[animation] = import\_folder(full\_path)  
  
 *def* animate(*self*, dt):  
 *self*.frame\_index += 9 \* dt  
 *if self*.frame\_index >= len(*self*.animations[*self*.status]):  
 *self*.frame\_index = 0  
  
 *self*.image = *self*.animations[*self*.status][int(*self*.frame\_index)]  
  
 *def* input(*self*):  
 keys = pygame.key.get\_pressed()  
  
 *if not self*.timers['axe use'].active *and not self*.sleep *and not self*.timers['water use'].active *and* \  
 *not self*.timers['hoe use'].active:  
 *# directions  
 if* keys[pygame.K\_w]:  
 *self*.status = 'up'  
 *self*.direction.y = -1  
 *elif* keys[pygame.K\_s]:  
 *self*.status = 'down'  
 *self*.direction.y = 1  
 *else*:  
 *self*.direction.y = 0  
  
 *if* keys[pygame.K\_d]:  
 *self*.status = 'right'  
 *self*.direction.x = 1  
 *elif* keys[pygame.K\_a]:  
 *self*.status = 'left'  
 *self*.direction.x = -1  
 *else*:  
 *self*.direction.x = 0  
  
 *if* keys[pygame.K\_LSHIFT]:  
 *self*.speed = 300  
 *if* keys[pygame.K\_d]:  
 *self*.status = 'right\_run'  
 *elif* keys[pygame.K\_a]:  
 *self*.status = 'left\_run'  
 *elif* keys[pygame.K\_w]:  
 *self*.status = 'up\_run'  
 *elif* keys[pygame.K\_s]:  
 *self*.status = 'down\_run'  
 *else*:  
 *self*.speed = 200  
  
 *# tool use  
 if* keys[pygame.K\_SPACE]:  
 *# timer for the tool use  
 if self*.selected\_tool == 'water':  
 *self*.timers['water use'].activate()  
 *elif self*.selected\_tool == 'hoe':  
 *self*.timers['hoe use'].activate()  
 *elif self*.selected\_tool == 'axe':  
 *self*.timers['axe use'].activate()  
 *self*.direction = pygame.math.Vector2()  
 *self*.frame\_index = 0  
 *self*.wave.play()  
  
 *# change tool  
 if not self*.timers['tool switch'].active:  
 *if* keys[pygame.K\_q] *or* keys[pygame.K\_e]:  
 *self*.switch.play()  
 *self*.timers['tool switch'].activate()  
 *if* keys[pygame.K\_q]:  
 *self*.tool\_index += 1  
 *self*.tool\_index = *self*.tool\_index *if self*.tool\_index < len(*self*.tools) *else* 0  
 *self*.selected\_tool = *self*.tools[*self*.tool\_index]  
 *else*:  
 *self*.tool\_index -= 1  
 *self*.tool\_index = *self*.tool\_index *if self*.tool\_index >= 0 *else* len(*self*.tools) - 1  
 *self*.selected\_tool = *self*.tools[*self*.tool\_index]  
  
 *# change seed  
 if not self*.timers['seed switch'].active:  
 *if* keys[pygame.K\_1]:  
 *self*.timers['seed switch'].activate()  
 *self*.switch.play()  
 *self*.seed\_select\_index = 0  
 *self*.seed\_index = 0  
 *if* keys[pygame.K\_2]:  
 *self*.timers['seed switch'].activate()  
 *self*.switch.play()  
 *self*.seed\_select\_index = 1  
 *self*.seed\_index = 1  
 *if* keys[pygame.K\_3]:  
 *self*.timers['seed switch'].activate()  
 *self*.switch.play()  
 *self*.seed\_select\_index = 2  
 *self*.seed\_index = 2  
 *if* keys[pygame.K\_4]:  
 *self*.timers['seed switch'].activate()  
 *self*.switch.play()  
 *self*.seed\_select\_index = 3  
 *self*.seed\_index = 3  
 *if* keys[pygame.K\_5]:  
 *self*.timers['seed switch'].activate()  
 *self*.switch.play()  
 *self*.seed\_select\_index = 4  
 *self*.seed\_index = 4  
 *if* keys[pygame.K\_6]:  
 *self*.timers['seed switch'].activate()  
 *self*.switch.play()  
 *self*.seed\_select\_index = 5  
 *self*.seed\_index = 5  
 *if* keys[pygame.K\_7]:  
 *self*.timers['seed switch'].activate()  
 *self*.switch.play()  
 *self*.seed\_select\_index = 6  
 *self*.seed\_index = 6  
 *if* keys[pygame.K\_8]:  
 *self*.timers['seed switch'].activate()  
 *self*.switch.play()  
 *self*.seed\_select\_index = 7  
 *self*.seed\_index = 7  
  
 *self*.selected\_seed = *self*.seeds[*self*.seed\_index]  
  
 *# seed use  
 if* keys[pygame.K\_TAB]:  
 *# timer for the seed use  
 self*.timers['seed use'].activate()  
 *self*.direction = pygame.math.Vector2()  
 *self*.frame\_index = 0  
  
 *# shop  
 if* keys[pygame.K\_f]:  
 collided\_interaction\_sprite = pygame.sprite.spritecollide(*self*, *self*.interaction, *False*)  
  
 *if* collided\_interaction\_sprite:  
 *if* collided\_interaction\_sprite[0].name == 'Trader':  
 *self*.toggle\_shop()  
  
 *if* collided\_interaction\_sprite[0].name == 'Bed':  
 *self*.status = 'left\_idle'  
 *self*.sleep = *True  
  
 # menu  
 if* keys[pygame.K\_ESCAPE]:  
 *self*.start\_menu()  
 *self*.choice.play()  
  
 *def* get\_status(*self*):  
 *# idle  
 # if the player is not moving:  
 if self*.direction.magnitude() == 0:  
 *# add \_idle to the status  
 self*.status = *self*.status.split('\_')[0] + '\_idle'  
  
 *# tool use  
 if self*.timers['axe use'].active *or self*.timers['water use'].active *or self*.timers['hoe use'].active:  
 *self*.status = *self*.status.split('\_')[0] + '\_' + *self*.selected\_tool  
  
 *def* update\_timer(*self*):  
 *for* timer *in self*.timers.values():  
 timer.update()  
  
 *def* collision(*self*, direction):  
 *for* sprite *in self*.collision\_sprite.sprites():  
 *if* hasattr(sprite, 'hitbox'):  
 *if* sprite.hitbox.colliderect(*self*.hitbox):  
 *if* direction == 'horizontal':  
 *if self*.direction.x > 0: *# moving right  
 self*.hitbox.right = sprite.hitbox.left  
 *if self*.direction.x < 0: *# moving left  
 self*.hitbox.left = sprite.hitbox.right  
 *self*.rect.centerx = *self*.hitbox.centerx  
 *self*.pos.x = *self*.hitbox.centerx  
 *if* direction == 'vertical':  
 *if self*.direction.y > 0: *# moving down  
 self*.hitbox.bottom = sprite.hitbox.top  
 *if self*.direction.y < 0: *# moving up  
 self*.hitbox.top = sprite.hitbox.bottom  
 *self*.rect.centery = *self*.hitbox.centery  
 *self*.pos.y = *self*.hitbox.centery  
  
 *def* move(*self*, dt):  
 *# normalizing a vector  
 if self*.direction.magnitude() > 0:  
 *self*.direction = *self*.direction.normalize()  
  
 *# horizontal movement  
 self*.pos.x += *self*.direction.x \* *self*.speed \* dt  
 *self*.hitbox.centerx = round(*self*.pos.x)  
 *self*.rect.centerx = *self*.hitbox.centerx  
 *self*.collision('horizontal')  
  
 *# vertical movement  
 self*.pos.y += *self*.direction.y \* *self*.speed \* dt  
 *self*.hitbox.centery = round(*self*.pos.y)  
 *self*.rect.centery = *self*.hitbox.centery  
 *self*.collision('vertical')  
  
 *def* update(*self*, dt):  
 *self*.input()  
 *self*.get\_status()  
 *self*.update\_timer()  
 *self*.get\_target\_pos()  
  
 *self*.move(dt)  
 *self*.animate(dt)  
  
 *self*.watering.set\_volume(SOUND\_VOLUME['Water'])  
 *self*.switch.set\_volume(SOUND\_VOLUME['Switch tool'])

**start\_menu.py**

*import* os  
*import* pygame  
*import* hashlib  
*from* settings *import* \*  
*from* timer *import* Timer  
  
  
*class* StartMenu:  
 *def \_\_init\_\_*(*self*, toggle):  
 *self*.toggle = toggle  
  
 *self*.accounts = {}  
 *self*.authorization\_status = ''  
 *self*.authorization = *False  
  
 # display  
 self*.display\_surface = pygame.display.get\_surface()  
  
 *# timer  
 self*.timer\_back = Timer(150)  
 *self*.timer\_down = Timer(150)  
  
 *# font  
 self*.font = pygame.font.Font('../font/Pixeltype.ttf', 50)  
 *self*.font\_enter = pygame.font.Font('../font/Pixeltype.ttf', 40)  
 *self*.font\_error = pygame.font.Font('../font/Pixeltype.ttf', 30)  
 *self*.font\_text = pygame.font.Font('../font/Pixeltype.ttf', 75)  
 *self*.font\_name = pygame.font.Font('../font/Pixeltype.ttf', 110)  
  
 *# imports* overlay\_path = '../graphics/start menu/'  
 *self*.slot\_surf = pygame.image.load(f'{overlay\_path}slot.png').convert\_alpha()  
 *self*.select\_slot\_surf = pygame.image.load(f'{overlay\_path}select\_slot.png').convert\_alpha()  
 *self*.enter\_slot\_surf = pygame.image.load(f'{overlay\_path}enter\_slot.png').convert\_alpha()  
 *self*.select\_enter\_slot\_surf = pygame.image.load(f'{overlay\_path}select\_enter\_slot.png').convert\_alpha()  
 *self*.background\_down\_surf = pygame.image.load(f'{overlay\_path}background\_down.png').convert\_alpha()  
 *self*.background\_up\_surf = pygame.image.load(f'{overlay\_path}background\_up.png').convert\_alpha()  
 *self*.play\_key\_surf = pygame.image.load(f'{overlay\_path}play\_up.png').convert\_alpha()  
 *self*.select\_play\_key\_surf = pygame.image.load(f'{overlay\_path}select\_play.png').convert\_alpha()  
  
 *# login and password  
 self*.login\_active = *False  
 self*.password\_active = *False  
  
 self*.index = 0  
 *self*.login\_text = ''  
 *self*.password\_text = ''  
 *self*.password\_text\_copy = ''  
  
 *# sound  
 self*.switch = pygame.mixer.Sound('../audio/switch.mp3')  
 *self*.switch.set\_volume(0.05)  
  
 *# keys  
 self*.keys = ['q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', 'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l',  
 'z', 'x', 'c', 'v', 'b', 'n', 'm', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'BACKSPACE']  
  
 *def* draw\_screen(*self*):  
 *if not self*.authorization:  
 *# draw background* background\_down\_rect = *self*.background\_down\_surf.get\_rect(center=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2))  
 *self*.display\_surface.blit(*self*.background\_down\_surf, background\_down\_rect)  
  
 background\_up\_rect = *self*.background\_up\_surf.get\_rect(center=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2))  
 *self*.display\_surface.blit(*self*.background\_up\_surf, background\_up\_rect)  
  
 *# game name* name\_surf = *self*.font\_name.render("Cat's Farm", *True*, "White")  
 login\_rect = name\_surf.get\_rect(midtop=(SCREEN\_WIDTH // 2, 100))  
 *self*.display\_surface.blit(name\_surf, (login\_rect[0] + 4, login\_rect[1] + 12))  
  
 *# login slot  
 self*.login\_surf = *self*.slot\_surf  
 *self*.login\_rect = *self*.login\_surf.get\_rect(midbottom=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2 - 10))  
 *self*.display\_surface.blit(*self*.login\_surf, *self*.login\_rect)  
  
 *# password slot  
 self*.password\_surf = *self*.slot\_surf  
 *self*.password\_rect = *self*.password\_surf.get\_rect(midbottom=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2 + 150))  
 *self*.display\_surface.blit(*self*.password\_surf, *self*.password\_rect)  
  
 *# select  
 if self*.login\_active *or self*.password\_active:  
 select\_surf = *self*.select\_slot\_surf  
 select\_rect = *self*.login\_surf.get\_rect(midbottom=(SCREEN\_WIDTH // 2 - 3, SCREEN\_HEIGHT // 2 - 13 + (*self*.index \* 160)))  
 *self*.display\_surface.blit(select\_surf, select\_rect)  
  
 *# text name* login\_surf = *self*.font\_text.render('Login', *True*, 'White')  
 login\_rect = login\_surf.get\_rect(midtop=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2 - 155))  
 *self*.display\_surface.blit(login\_surf, (login\_rect[0], login\_rect[1] + 5))  
  
 password\_surf = *self*.font\_text.render('Password', *True*, 'White')  
 password\_rect = password\_surf.get\_rect(midtop=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2 + 5))  
 *self*.display\_surface.blit(password\_surf, (password\_rect[0], password\_rect[1] + 5))  
  
 *# login and password text input* login\_text\_surf = *self*.font.render(*self*.login\_text, *True*, 'black')  
 login\_text\_rect = login\_text\_surf.get\_rect(center=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2 - 60))  
 *self*.display\_surface.blit(login\_text\_surf, (login\_text\_rect[0], login\_text\_rect[1] + 9))  
  
 password\_text\_surf = *self*.font.render(*self*.password\_text, *True*, 'black')  
 password\_text\_rect = password\_text\_surf.get\_rect(center=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2 + 100))  
 *self*.display\_surface.blit(password\_text\_surf, (password\_text\_rect[0], password\_text\_rect[1] + 9))  
  
 *# authorization and registration  
 self*.entry\_surf = *self*.enter\_slot\_surf  
 *self*.entry\_rect = *self*.entry\_surf.get\_rect(center=(SCREEN\_WIDTH // 2 - 100, SCREEN\_HEIGHT - 117))  
 *self*.display\_surface.blit(*self*.entry\_surf, *self*.entry\_rect)  
  
 authorization\_text\_surf = *self*.font\_enter.render('Enter', *True*, '#b68962')  
 authorization\_text\_rect = authorization\_text\_surf.get\_rect(center=(SCREEN\_WIDTH // 2 - 95, SCREEN\_HEIGHT - 125))  
 *self*.display\_surface.blit(authorization\_text\_surf, (authorization\_text\_rect[0], authorization\_text\_rect[1] + 9))  
  
 *self*.registration\_surf = *self*.enter\_slot\_surf  
 *self*.registration\_rect = *self*.registration\_surf.get\_rect(center=(SCREEN\_WIDTH // 2 + 105, SCREEN\_HEIGHT -117))  
 *self*.display\_surface.blit(*self*.registration\_surf, *self*.registration\_rect)  
  
 registration\_text\_surf = *self*.font\_enter.render('Registration', *True*, '#b68962')  
 registration\_text\_rect = registration\_text\_surf.get\_rect(center=(SCREEN\_WIDTH // 2 + 108, SCREEN\_HEIGHT - 125))  
 *self*.display\_surface.blit(registration\_text\_surf, (registration\_text\_rect[0], registration\_text\_rect[1] + 9))  
  
 *# authorization error  
 if self*.authorization\_status != '':  
 *if self*.authorization\_status == 'invalid':  
 error\_text\_surf = *self*.font\_error.render('Invalid username or password', *True*, '#f24646')  
  
 *elif self*.authorization\_status == 'none':  
 error\_text\_surf = *self*.font\_error.render('There is no such account', *True*, '#f24646')  
  
 *elif self*.authorization\_status == 'true':  
 error\_text\_surf = *self*.font\_error.render('Successfully registered!', *True*, '#d2e077')  
  
 *elif self*.authorization\_status == 'empty':  
 error\_text\_surf = *self*.font\_error.render('Fill in the empty fields', *True*, '#f24646')  
  
 *elif self*.authorization\_status == 'exists':  
 error\_text\_surf = *self*.font\_error.render('An account with this username already exists', *True*,  
 '#f24646')  
  
 error\_text\_rect = error\_text\_surf.get\_rect(center=(SCREEN\_WIDTH // 2 + 3, SCREEN\_HEIGHT - 175))  
 *self*.display\_surface.blit(error\_text\_surf, error\_text\_rect)  
  
 *elif self*.authorization:  
 *# background* background\_down\_rect = *self*.background\_down\_surf.get\_rect(center=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2))  
 *self*.display\_surface.blit(*self*.background\_down\_surf, background\_down\_rect)  
  
 *# play key  
 self*.play\_key\_rect = *self*.play\_key\_surf.get\_rect(center=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT - 350))  
 *self*.display\_surface.blit(*self*.play\_key\_surf, *self*.play\_key\_rect)  
  
 *def* input(*self*):  
 *self*.timer\_back.update()  
 *self*.timer\_down.update()  
 keys = pygame.key.get\_pressed()  
 mouse = pygame.mouse.get\_pressed()[0]  
 mouse\_pos = pygame.mouse.get\_pos()  
  
 *if not self*.authorization:  
 *if self*.entry\_rect.collidepoint(mouse\_pos):  
 select\_enter\_slot\_rect = *self*.select\_enter\_slot\_surf.get\_rect(center=(SCREEN\_WIDTH // 2 - 100, SCREEN\_HEIGHT - 117))  
 *self*.display\_surface.blit(*self*.select\_enter\_slot\_surf, select\_enter\_slot\_rect)  
 *elif self*.registration\_rect.collidepoint(mouse\_pos):  
 select\_enter\_slot\_rect = *self*.select\_enter\_slot\_surf.get\_rect(center=(SCREEN\_WIDTH // 2 + 105, SCREEN\_HEIGHT - 117))  
 *self*.display\_surface.blit(*self*.select\_enter\_slot\_surf, select\_enter\_slot\_rect)  
 *else*:  
 *if self*.play\_key\_rect.collidepoint(mouse\_pos):  
 select\_play\_rect = *self*.select\_play\_key\_surf.get\_rect(center=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT - 350))  
 *self*.display\_surface.blit(*self*.select\_play\_key\_surf, select\_play\_rect)  
  
 *# mouse click  
 if not self*.timer\_down.active *and not self*.timer\_back.active:  
 *if* mouse:  
 *if self*.login\_rect.collidepoint(mouse\_pos):  
 *self*.login\_active = *True  
 self*.password\_active = *False  
  
 self*.index = 0  
  
 *elif self*.password\_rect.collidepoint(mouse\_pos):  
 *self*.login\_active = *False  
 self*.password\_active = *True  
  
 self*.index = 1  
  
 *if self*.authorization:  
 *if self*.play\_key\_rect.collidepoint(mouse\_pos):  
 *self*.toggle()  
  
 *if self*.entry\_rect.collidepoint(mouse\_pos):  
 status = *self*.check\_account()  
 *if* status == 'entry':  
 *self*.authorization = *True  
 if* status == 'empty':  
 *self*.authorization\_status = 'empty'  
 *if* status == 'invalid':  
 *self*.authorization\_status = 'invalid'  
 *if* status == 'none':  
 *self*.authorization\_status = 'none'  
 *self*.timer\_back.activate()  
  
 *elif self*.registration\_rect.collidepoint(mouse\_pos):  
 status = *self*.registrate()  
 *if* status == 'true':  
 *self*.authorization\_status = 'true'  
 *if* status == 'empty':  
 *self*.authorization\_status = 'empty'  
 *if* status == 'exists':  
 *self*.authorization\_status = 'exists'  
 *self*.timer\_back.activate()  
  
 *# keys down  
 if* any(keys):  
 *if not self*.timer\_down.active *and not self*.timer\_back.active:  
 *for* i *in self*.keys:  
 key\_backspace = pygame.key.key\_code('BACKSPACE')  
 key\_name = pygame.key.key\_code(i)  
  
 *if self*.login\_active:  
  
 *if* keys[key\_backspace]:  
 *self*.login\_text = *self*.login\_text[:-1]  
 *self*.timer\_back.activate()  
 *break  
 elif* keys[key\_name]:  
 *if* len(*self*.login\_text) < 14:  
 *self*.timer\_down.activate()  
 *self*.login\_text += i  
 *break  
  
 if self*.password\_active:  
  
 *if* keys[key\_backspace]:  
 *self*.password\_text = *self*.password\_text[:-1]  
 *self*.password\_text\_copy = *self*.password\_text\_copy[:-1]  
 *self*.timer\_back.activate()  
 *break  
 elif* keys[key\_name]:  
 *if* len(*self*.password\_text) < 14:  
 *self*.timer\_down.activate()  
 *self*.password\_text\_copy += i  
 *self*.password\_text += '#'  
 *break  
  
 def* check\_account(*self*):  
 *if* len(*self*.login\_text) == 0 *or* len(*self*.password\_text\_copy) == 0:  
 *return* 'empty'  
 *else*:  
 password\_code = hashlib.sha256(*self*.password\_text\_copy.encode()).hexdigest()  
 file = open('../save/accounts.txt', 'r+')  
 a = file.readline()[:-1].split(' ')  
  
 *while True*:  
 *if* a != ['']:  
 *self*.accounts[a[0]] = a[1]  
 a = file.readline()[:-1].split(' ')  
 *else*:  
 *break* true\_account = *False* account\_exists = *False  
 for* i *in self*.accounts.items():  
 login, password = i  
 *if self*.login\_text == login *and* password\_code == password:  
 true\_account = *True  
 break  
 if self*.login\_text == login *and* password\_code != password:  
 account\_exists = *True  
 break  
  
 if* true\_account:  
 *return* 'entry'  
 *elif* account\_exists:  
 *return* 'invalid'  
 *elif not* true\_account *and not* account\_exists:  
 *return* 'none'  
  
 *def* registrate(*self*):  
 *if* len(*self*.login\_text) == 0 *or* len(*self*.password\_text\_copy) == 0:  
 *return* 'empty'  
 *else*:  
 password\_code = hashlib.sha256(*self*.password\_text\_copy.encode()).hexdigest()  
 file = open('../save/accounts.txt', 'r+')  
 a = file.readline()[:-1].split(' ')  
  
 *while True*:  
 *if* a != ['']:  
 *self*.accounts[a[0]] = a[1]  
 a = file.readline()[:-1].split(' ')  
 *else*:  
 *break* account\_exists = *False  
  
 for* i *in self*.accounts.items():  
 login, password = i  
 *if self*.login\_text == login:  
 account\_exists = *True* print('true')  
  
 *if not* account\_exists:  
 file = open('../save/accounts.txt', 'r+')  
 file.seek(0, os.SEEK\_END)  
 file.write(f'{*self*.login\_text} {password\_code}\n')  
 file.close()  
 *return* 'true'  
 *else*:  
 *return* 'exists'  
  
 *def* update(*self*):  
 *self*.draw\_screen()  
 *self*.input()

**support.py**

*import* pygame  
*from* os *import* walk  
  
  
*def* import\_folder(path):  
 surface\_list = []  
  
 *for* \_, \_\_, img\_files *in* walk(path):  
 *for* image *in* img\_files:  
 full\_path = path + '/' + image  
 image\_surf = pygame.image.load(full\_path).convert\_alpha()  
 surface\_list.append(image\_surf)  
  
 *return* surface\_list  
  
  
*def* import\_folder\_dict(path):  
 surface\_dict = {}  
  
 *for* \_, \_\_, img\_files *in* walk(path):  
 *for* image *in* img\_files:  
 full\_path = path + '/' + image  
 image\_surf = pygame.image.load(full\_path).convert\_alpha()  
 surface\_dict[image.split('.')[0]] = image\_surf  
  
 *return* surface\_dict

**timer.py**

*import* pygame  
  
  
*class* Timer:  
 *def \_\_init\_\_*(*self*, duration, func=*None*):  
 *self*.duration = duration  
 *self*.func = func  
 *self*.start\_time = 0  
 *self*.active = *False  
  
 def* activate(*self*):  
 *self*.active = *True  
 self*.start\_time = pygame.time.get\_ticks()  
  
 *def* deactivate(*self*):  
 *self*.active = *False  
 self*.start\_time = 0  
  
 *def* update(*self*):  
 current\_time = pygame.time.get\_ticks()  
 *if* current\_time - *self*.start\_time >= *self*.duration:  
 *if self*.func *and self*.start\_time != 0:  
 *self*.func()  
 *self*.deactivate()

**transition.py**

*import* pygame  
*from* settings *import* \*  
  
  
*class* Transition:  
 *def \_\_init\_\_*(*self*, reset, player):  
 *# setup  
 self*.display\_surface = pygame.display.get\_surface()  
 *self*.reset = reset  
 *self*.player = player  
  
 *# overlay image  
 self*.image = pygame.Surface((SCREEN\_WIDTH, SCREEN\_HEIGHT))  
 *self*.color = 255  
 *self*.speed = -3  
  
 *def* play(*self*):  
 *self*.color += *self*.speed  
 *if self*.color <= 0:  
 *self*.speed \*= -1  
 *self*.color = 0  
 *self*.reset()  
  
 *if self*.color > 255:  
 *self*.color = 255  
 *self*.player.sleep = *False  
 self*.speed = -3  
  
 *self*.image.fill((*self*.color, *self*.color, *self*.color))  
 *self*.display\_surface.blit(*self*.image, (0, 0), special\_flags=pygame.BLEND\_RGBA\_MULT)

**settings.py**

*import* pygame  
*from* settings *import* \*  
  
  
*class* Transition:  
 *def \_\_init\_\_*(*self*, reset, player):  
 *# setup  
 self*.display\_surface = pygame.display.get\_surface()  
 *self*.reset = reset  
 *self*.player = player  
  
 *# overlay image  
 self*.image = pygame.Surface((SCREEN\_WIDTH, SCREEN\_HEIGHT))  
 *self*.color = 255  
 *self*.speed = -3  
  
 *def* play(*self*):  
 *self*.color += *self*.speed  
 *if self*.color <= 0:  
 *self*.speed \*= -1  
 *self*.color = 0  
 *self*.reset()  
  
 *if self*.color > 255:  
 *self*.color = 255  
 *self*.player.sleep = *False  
 self*.speed = -3  
  
 *self*.image.fill((*self*.color, *self*.color, *self*.color))  
 *self*.display\_surface.blit(*self*.image, (0, 0), special\_flags=pygame.BLEND\_RGBA\_MULT)

**overlay.py**

*import* pygame  
*from* settings *import* \*  
  
  
*class* Overlay:  
 *def \_\_init\_\_*(*self*, player):  
 *# general setup  
 self*.display\_surface = pygame.display.get\_surface()  
 *self*.player = player  
  
 *# time  
 self*.clock = pygame.time.get\_ticks()  
  
 *# imports  
 self*.font = pygame.font.Font('../font/Pixeltype.ttf', 37)  
 *self*.font\_seed = pygame.font.Font('../font/Pixeltype.ttf', 25)  
 *self*.font\_time = pygame.font.Font('../font/Pixeltype.ttf', 40)  
 overlay\_path = '../graphics/overlay/'  
 *self*.tools\_sufr = {tool: pygame.image.load(f'{overlay\_path}{tool}.png').convert\_alpha() *for* tool *in* player.tools}  
 *self*.seeds\_sufr = {seed: pygame.image.load(f'{overlay\_path}{seed}.png').convert\_alpha() *for* seed *in* player.seeds}  
 *self*.slot\_surf = pygame.image.load(f'{overlay\_path}slot.png').convert\_alpha()  
 *self*.background\_surf = pygame.image.load(f'{overlay\_path}back.png').convert\_alpha()  
 *self*.hp\_money\_bar = pygame.image.load(f'{overlay\_path}hp\_money\_bar.png').convert\_alpha()  
 *self*.select\_slot = pygame.image.load(f'{overlay\_path}select.png').convert\_alpha()  
  
 *def* display(*self*):  
 *# tools* tool\_surf = *self*.tools\_sufr[*self*.player.selected\_tool]  
 tool\_rect = tool\_surf.get\_rect(center=OVERLAY\_POSITIONS['tool'])  
  
 *self*.display\_surface.blit(tool\_surf, tool\_rect)  
  
 *# hp and money bar* hp\_money\_bar\_rect = *self*.hp\_money\_bar.get\_rect(topleft=OVERLAY\_POSITIONS['hp\_money\_bar'])  
 *self*.display\_surface.blit(*self*.hp\_money\_bar, hp\_money\_bar\_rect)  
  
 text\_surf = *self*.font.render(f'{*self*.player.money}', *True*, '#b68962')  
 text\_rect = text\_surf.get\_rect(topleft=(48, 118))  
 *self*.display\_surface.blit(text\_surf, (text\_rect[0], text\_rect[1] + 5))  
  
 *# background* back\_rect = *self*.slot\_surf.get\_rect(midbottom=OVERLAY\_POSITIONS['background'])  
 *self*.display\_surface.blit(*self*.background\_surf, back\_rect)  
  
 *# slot  
 for* i *in* range(9):  
 slot\_rect = *self*.slot\_surf.get\_rect(center=((390 + (i \* 62)), SCREEN\_HEIGHT - 59))  
 *self*.display\_surface.blit(*self*.slot\_surf, slot\_rect)  
  
 *# select slot* select\_slot\_rect = *self*.select\_slot.get\_rect(  
 center=((390 + (*self*.player.seed\_select\_index \* 62)), SCREEN\_HEIGHT - 59))  
 *self*.display\_surface.blit(*self*.select\_slot, select\_slot\_rect)  
  
 *# seeds  
 for* i *in* range(len(*self*.seeds\_sufr)):  
 seed\_surf = list(*self*.seeds\_sufr.values())[i]  
 seed\_rect = seed\_surf.get\_rect(center=((389 + (62 \* i)), SCREEN\_HEIGHT - 60))  
  
 text\_surf = *self*.font\_seed.render(f'{*self*.player.seed\_inventory[*self*.player.seeds[i]]}', *True*, 'black')  
 text\_rect = text\_surf.get\_rect(topleft=(seed\_rect.centerx+16, seed\_rect.centery+16))  
  
 *self*.display\_surface.blit(seed\_surf, seed\_rect)  
 *self*.display\_surface.blit(text\_surf, text\_rect)  
  
 *# time  
 self*.clock = pygame.time.get\_ticks() // 1000  
 text\_surf = *self*.font\_time.render(f'{*self*.clock//60+6}: {"0" + str(*self*.clock % 60) *if self*.clock % 60 < 10 *else self*.clock - (*self*.clock//60\*60)}', *True*, '#b68962')  
 text\_rect = text\_surf.get\_rect(topright=(SCREEN\_WIDTH-30, 20))  
  
 *self*.display\_surface.blit(text\_surf, text\_rect)

**shop.py**

*import* pygame  
*from* settings *import* \*  
*from* timer *import* Timer  
  
  
*class* Shop:  
 *def \_\_init\_\_*(*self*, player, toggle\_menu):  
 *# general setup  
 self*.player = player  
 *self*.toggle\_menu = toggle\_menu  
 *self*.display\_surface = pygame.display.get\_surface()  
 *self*.font = pygame.font.Font('../font/Pixeltype.ttf', 50)  
  
 *# options  
 self*.width = 400  
 *self*.space = 10  
 *self*.padding = 8  
  
 *# entries  
 self*.options = list(*self*.player.item\_inventory.keys()) + list(*self*.player.seed\_inventory.keys())  
 *self*.sell\_border = len(*self*.player.item\_inventory) - 1  
 *self*.setup()  
  
 *# movement  
 self*.index = 0  
 *self*.timer = Timer(200)  
  
 *# sound  
 self*.buy\_sell = pygame.mixer.Sound('../audio/buy or sell.mp3')  
 *self*.buy\_sell.set\_volume(SOUND\_VOLUME['Bye or Sell'])  
  
 *def* display\_money(*self*):  
 text\_surf = *self*.font.render(f'${*self*.player.money}', *True*, '#b68962')  
 text\_rect = text\_surf.get\_rect(midbottom=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT - 20))  
  
 pygame.draw.rect(*self*.display\_surface, '#dcb98a', text\_rect.inflate(10, 10), 0, 4)  
 *self*.display\_surface.blit(text\_surf, (text\_rect[0], text\_rect[1] + 5))  
  
 *def* setup(*self*):  
 *# create text surfaces  
 self*.text\_surfs = []  
 *self*.total\_height = 0  
 *for* item *in self*.options:  
 text\_surf = *self*.font.render(item, *True*, 'Black')  
 *self*.text\_surfs.append(text\_surf)  
 *self*.total\_height += text\_surf.get\_height() + (*self*.padding \* 2)  
  
 *self*.total\_height += (len(*self*.text\_surfs) - 1) \* *self*.space  
 *self*.menu\_top = SCREEN\_HEIGHT // 2 - *self*.total\_height // 2  
 *self*.main\_rect = pygame.Rect(SCREEN\_WIDTH // 2 - *self*.width // 2, *self*.menu\_top, *self*.width, *self*.total\_height)  
  
 *# buy / sell text surface  
 self*.buy\_text = *self*.font.render('buy', *True*, 'Black')  
 *self*.sell\_text = *self*.font.render('sell', *True*, 'Black')  
  
 *def* input(*self*):  
 keys = pygame.key.get\_pressed()  
 *self*.timer.update()  
  
 *if not self*.timer.active:  
 *if* keys[pygame.K\_w]:  
 *self*.index -= 1  
 *self*.timer.activate()  
  
 *if* keys[pygame.K\_s]:  
 *self*.index += 1  
 *self*.timer.activate()  
  
 *if* keys[pygame.K\_BACKSPACE]:  
 *self*.player.toggle\_shop()  
  
 *if* keys[pygame.K\_SPACE]:  
 *self*.timer.activate()  
  
 *# get item* current\_item = *self*.options[*self*.index]  
  
 *# sell  
 if self*.index <= *self*.sell\_border:  
 *if self*.player.item\_inventory[current\_item] > 0:  
 *self*.player.item\_inventory[current\_item] -= 1  
 *self*.player.money += SALE\_PRICES[current\_item]  
 *self*.buy\_sell.stop()  
 *self*.buy\_sell.play()  
  
 *# buy  
 else*:  
 *if self*.player.money > PURCHASE\_PRICES[current\_item]:  
 *self*.player.seed\_inventory[current\_item] += 1  
 *self*.player.money -= PURCHASE\_PRICES[current\_item]  
 *self*.buy\_sell.stop()  
 *self*.buy\_sell.play()  
  
 *# clamp the values  
 if self*.index < 0:  
 *self*.index = len(*self*.text\_surfs) - 1  
 *if self*.index > len(*self*.text\_surfs) - 1:  
 *self*.index = 0  
  
 *def* show\_entry(*self*, text\_surf, amount, top, selected):  
 *# background* bg\_rect = pygame.Rect(*self*.main\_rect.left, top, *self*.width, text\_surf.get\_height() + (*self*.padding \* 2))  
 pygame.draw.rect(*self*.display\_surface, 'White', bg\_rect, 0, 4)  
  
 *# text* text\_rect = text\_surf.get\_rect(midleft=(*self*.main\_rect.left + 20, bg\_rect.centery + 3))  
 *self*.display\_surface.blit(text\_surf, text\_rect)  
  
 *# amount* amount\_surf = *self*.font.render(str(amount), *True*, 'Black')  
 amount\_rect = amount\_surf.get\_rect(midright=(*self*.main\_rect.right - 20, bg\_rect.centery + 4))  
 *self*.display\_surface.blit(amount\_surf, amount\_rect)  
  
 *# selected  
 if* selected:  
 pygame.draw.rect(*self*.display\_surface, 'black', bg\_rect, 4, 4)  
 *if self*.index <= *self*.sell\_border: *# sell* pos\_rect = *self*.sell\_text.get\_rect(midleft=(*self*.main\_rect.left + 180, bg\_rect.centery + 3))  
 *self*.display\_surface.blit(*self*.sell\_text, pos\_rect)  
 *else*: *# buy* pos\_rect = *self*.buy\_text.get\_rect(midleft=(*self*.main\_rect.left + 180, bg\_rect.centery + 3))  
 *self*.display\_surface.blit(*self*.buy\_text, pos\_rect)  
  
 *def* update(*self*):  
 *self*.input()  
 *self*.display\_money()  
  
 *for* text\_index, text\_surf *in* enumerate(*self*.text\_surfs):  
 top = *self*.main\_rect.top + text\_index \* (text\_surf.get\_height() + (*self*.padding \* 2) + *self*.space)  
  
 amount\_list = list(*self*.player.item\_inventory.values()) + list(*self*.player.seed\_inventory.values())  
 amount = amount\_list[text\_index]  
  
 *self*.show\_entry(text\_surf, amount, top, *self*.index == text\_index)  
  
 *self*.buy\_sell.set\_volume(SOUND\_VOLUME['Bye or Sell'])

**soil.py**

*from* support *import* \*  
*from* settings *import* \*  
*from* random *import* choice  
*from* pytmx.util\_pygame *import* load\_pygame  
  
  
*class* SoilTile(pygame.sprite.Sprite):  
 *def \_\_init\_\_*(*self*, pos, surf, groups):  
 super().*\_\_init\_\_*(groups)  
  
 *self*.image = surf  
 *self*.rect = *self*.image.get\_rect(topleft=pos)  
 *self*.z = LAYERS['soil']  
  
  
*class* WaterTile(pygame.sprite.Sprite):  
 *def \_\_init\_\_*(*self*, pos, surf, groups):  
 super().\_\_init\_\_(groups)  
  
 *self*.image = surf  
 *self*.rect = *self*.image.get\_rect(topleft=pos)  
 *self*.z = LAYERS['soil water']  
  
  
*class* Plant(pygame.sprite.Sprite):  
 *def \_\_init\_\_*(*self*, plant\_type, groups, soil, check\_watered):  
 super().\_\_init\_\_(groups)  
  
 *# setup  
 self*.plant\_type = plant\_type  
 *self*.frames = import\_folder(f'../graphics/fruit/{plant\_type}')  
 *self*.soil = soil  
 *self*.check\_watered = check\_watered  
  
 *# plant growing  
 self*.age = 0  
 *self*.max\_age = len(*self*.frames) - 1  
 *self*.grow\_speed = GROW\_SPEED[plant\_type]  
 *self*.harvestable = *False  
  
 # sprite setup  
 self*.image = *self*.frames[*self*.age]  
 *self*.y\_offset = -25 *if* plant\_type == 'corn' *else* -30  
 *self*.rect = *self*.image.get\_rect(midbottom=*self*.soil.rect.midbottom + pygame.math.Vector2(0, *self*.y\_offset))  
 *self*.z = LAYERS['ground plant']  
  
 *def* grow(*self*):  
 *if self*.check\_watered(*self*.rect.center):  
 *self*.age += *self*.grow\_speed  
  
 *if* int(*self*.age) > 0:  
 *self*.z = LAYERS['main']  
 *self*.hitbox = *self*.image.get\_rect(midbottom=(*self*.rect.midbottom[0]-1, *self*.rect.midbottom[1]-27)).inflate(-*self*.rect.width \* 0.3, - *self*.rect.height \* 0.95)  
  
 *if self*.age >= *self*.max\_age:  
 *self*.age = *self*.max\_age  
 *self*.harvestable = *True  
  
 self*.image = *self*.frames[int(*self*.age)]  
 *self*.rect = *self*.image.get\_rect(midbottom=*self*.soil.rect.midbottom + pygame.math.Vector2(0, *self*.y\_offset))  
  
  
*class* SoilLayer:  
 *def \_\_init\_\_*(*self*, all\_sprites, collision\_sprites):  
 *# sprite groups  
 self*.all\_sprites = all\_sprites  
 *self*.collision\_sprites = collision\_sprites  
 *self*.soil\_sprites = pygame.sprite.Group()  
 *self*.water\_sprites = pygame.sprite.Group()  
 *self*.plant\_sprites = pygame.sprite.Group()  
  
 *# graphics  
 self*.soil\_surfs = pygame.image.load('../graphics/soil/x.png').convert\_alpha()  
 *self*.water\_surfs = import\_folder('../graphics/soil\_water/')  
  
 *self*.create\_soil\_grid()  
 *self*.create\_hit\_rects()  
  
 *# sounds  
 self*.hoe\_sound = pygame.mixer.Sound('../audio/hoe.mp3')  
 *self*.hoe\_sound.set\_volume(SOUND\_VOLUME['Hoe'])  
  
 *self*.plant\_sound = pygame.mixer.Sound('../audio/boarding3.mp3')  
 *self*.plant\_sound.set\_volume(SOUND\_VOLUME['Plant'])  
  
 *def* create\_soil\_grid(*self*):  
 ground = pygame.image.load('../graphics/world/ground.png')  
 h\_tiles, w\_tiles = ground.get\_width() // TILE\_SIZE, ground.get\_height() // TILE\_SIZE  
  
 *self*.grid = [[[] *for* \_ *in* range(h\_tiles)] *for* \_ *in* range(w\_tiles)]  
  
 *for* x, y, \_ *in* load\_pygame('../data/map.tmx').get\_layer\_by\_name('Farmable').tiles():  
 *self*.grid[y][x].append('F')  
  
 *def* create\_hit\_rects(*self*):  
 *self*.hit\_rects = []  
 *for* index\_row, row *in* enumerate(*self*.grid):  
 *for* index\_col, cell *in* enumerate(row):  
 *if* 'F' *in* cell:  
 x = index\_col \* TILE\_SIZE  
 y = index\_row \* TILE\_SIZE  
 rect = pygame.Rect(x, y, TILE\_SIZE, TILE\_SIZE)  
 *self*.hit\_rects.append(rect)  
  
 *def* get\_hit(*self*, point):  
 *for* rect *in self*.hit\_rects:  
 *if* rect.collidepoint(point):  
  
 x = rect.x // TILE\_SIZE  
 y = rect.y // TILE\_SIZE  
  
 *if* 'F' *in self*.grid[y][x] *and* 'X' *not in self*.grid[y][x]:  
 *# sound play  
 self*.hoe\_sound.play()  
  
 *# create soil  
 self*.grid[y][x].append('X')  
 *self*.create\_soil\_tiles()  
 *if self*.raining:  
 *self*.water\_all()  
  
 *def* water(*self*, target\_pos):  
 *for* soil\_sprites *in self*.soil\_sprites.sprites():  
 *if* soil\_sprites.rect.collidepoint(target\_pos):  
 x = soil\_sprites.rect.x // TILE\_SIZE  
 y = soil\_sprites.rect.y // TILE\_SIZE  
  
 *if* 'W' *not in self*.grid[y][x]:  
 WaterTile(pos=soil\_sprites.rect.topleft,  
 surf=choice(*self*.water\_surfs),  
 groups=[*self*.all\_sprites, *self*.water\_sprites])  
  
 *self*.grid[y][x].append('W')  
  
 *def* water\_all(*self*):  
 *for* index\_row, row *in* enumerate(*self*.grid):  
 *for* index\_col, cell *in* enumerate(row):  
 *if* 'X' *in* cell *and* 'W' *not in* cell:  
 cell.append('W')  
  
 WaterTile(pos=(index\_col \* TILE\_SIZE, index\_row \* TILE\_SIZE),  
 surf=choice(*self*.water\_surfs),  
 groups=[*self*.all\_sprites, *self*.water\_sprites])  
  
 *def* remove\_water(*self*):  
 *# destroy all water sprites  
 for* sprite *in self*.water\_sprites.sprites():  
 sprite.kill()  
  
 *# clean up the grid  
 for* row *in self*.grid:  
 *for* cell *in* row:  
 *if* 'W' *in* cell:  
 cell.remove('W')  
  
 *def* remove\_soil(*self*):  
 *for* soil\_sprite *in self*.soil\_sprites.sprites():  
  
 x = soil\_sprite.rect.x // TILE\_SIZE  
 y = soil\_sprite.rect.y // TILE\_SIZE  
  
 *if* 'P' *not in self*.grid[y][x]:  
 *self*.grid[y][x].remove('X')  
  
 *for* sprite *in self*.soil\_sprites.sprites():  
 sprite.kill()  
  
 *def* check\_watered(*self*, pos):  
 x = pos[0] // TILE\_SIZE  
 y = pos[1] // TILE\_SIZE  
 cell = *self*.grid[y][x]  
 is\_watered = 'W' *in* cell  
  
 *return* is\_watered  
  
 *def* plant\_seed(*self*, target\_pos, seed, seed\_inventory, selected\_seed):  
 *for* soil\_sprite *in self*.soil\_sprites.sprites():  
 *if* soil\_sprite.rect.collidepoint(target\_pos):  
  
 x = soil\_sprite.rect.x // TILE\_SIZE  
 y = soil\_sprite.rect.y // TILE\_SIZE  
  
 *if* 'P' *not in self*.grid[y][x]:  
 *self*.plant\_sound.play()  
  
 *self*.grid[y][x].append('P')  
  
 seed\_inventory[selected\_seed] -= 1  
  
 Plant(plant\_type=seed,  
 groups=[*self*.all\_sprites, *self*.plant\_sprites, *self*.collision\_sprites],  
 soil=soil\_sprite,  
 check\_watered=*self*.check\_watered)  
  
 *def* update\_plants(*self*):  
 *for* plant *in self*.plant\_sprites.sprites():  
 plant.grow()  
  
 *def* create\_soil\_tiles(*self*):  
 *self*.soil\_sprites.empty()  
 *for* index\_row, row *in* enumerate(*self*.grid):  
 *for* index\_col, cell *in* enumerate(row):  
 *if* 'X' *in* cell:  
 SoilTile(pos=(index\_col \* TILE\_SIZE, index\_row \* TILE\_SIZE),  
 surf=*self*.soil\_surfs,  
 groups=[*self*.all\_sprites, *self*.soil\_sprites])

**sprites.py**

*import* pygame  
*from* settings *import* \*  
*from* random *import* randint, choice  
  
  
*class* Generic(pygame.sprite.Sprite):  
 *def \_\_init\_\_*(*self*, pos, surf, groups, z=LAYERS['main']):  
 super().\_\_init\_\_(groups)  
  
 *self*.image = surf  
 *self*.rect = *self*.image.get\_rect(topleft=pos)  
 *self*.z = z  
 *self*.hitbox = *self*.image.get\_rect(topleft=(pos[0], pos[1]+10)).inflate(-*self*.rect.width \* 0.2, -*self*.rect.height \* 0.85)  
  
  
*class* Collision(Generic):  
 *def \_\_init\_\_*(*self*, name, pos, surf, groups):  
 super().*\_\_init\_\_*(pos, surf, groups)  
  
 *if* name == 'Collision':  
 *self*.hitbox = *self*.image.get\_rect(topleft=(pos[0], pos[1]-32)).inflate(-*self*.rect.width \* 0.04, -*self*.rect.height \* 0.13)  
 *if* name == 'Collision up':  
 *self*.hitbox = *self*.image.get\_rect(topleft=(pos[0], pos[1]-50)).inflate(0, -*self*.rect.height \* 0.7)  
 *if* name == 'Collision down':  
 *self*.hitbox = *self*.image.get\_rect(topleft=(pos[0], pos[1]-20)).inflate(0, -*self*.rect.height \* 0.5)  
 *if* name == 'Collision corner':  
 *self*.hitbox = *self*.image.get\_rect(topleft=(pos[0], pos[1]-32)).inflate(-*self*.rect.width \* 0.04, -*self*.rect.height \* 0.13)  
 *if* name == 'Collision left':  
 *self*.hitbox = *self*.image.get\_rect(topleft=(pos[0]-18, pos[1]-20)).inflate(-*self*.rect.width \* 0.6, -*self*.rect.height \* 0.5)  
 *if* name == 'Collision right':  
 *self*.hitbox = *self*.image.get\_rect(topleft=(pos[0]+18, pos[1]-20)).inflate(-*self*.rect.width \* 0.6, -*self*.rect.height \* 0.5)  
  
  
*class* Interaction(Generic):  
 *def \_\_init\_\_*(*self*, pos, size, groups, name):  
 surf = pygame.Surface(size)  
 super().\_\_init\_\_(pos, surf, groups)  
  
 *self*.name = name  
  
  
*class* WildFlower(Generic):  
 *def \_\_init\_\_*(*self*, pos, surf, groups):  
 super().*\_\_init\_\_*(pos, surf, groups)  
  
 *self*.hitbox = *self*.image.get\_rect(topleft=(pos[0], pos[1]-25)).inflate(-20, -*self*.rect.height \* 0.6)  
  
  
*class* Particle(Generic):  
 *def \_\_init\_\_*(*self*, pos, surf, groups, z, duration=200):  
 super().\_\_init\_\_(pos, surf, groups, z)  
  
 *self*.start\_time = pygame.time.get\_ticks()  
 *self*.duration = duration  
  
 *# white surface* mask\_surf = pygame.mask.from\_surface(*self*.image)  
 new\_surf = mask\_surf.to\_surface()  
 new\_surf.set\_colorkey((0, 0, 0))  
 *self*.image = new\_surf  
  
 *def* update(*self*, dt):  
 current\_time = pygame.time.get\_ticks()  
 *if* current\_time - *self*.start\_time > *self*.duration:  
 *self*.kill()  
  
  
*class* Tree(Generic):  
 *def \_\_init\_\_*(*self*, pos, surf, groups, name, player\_add):  
 super().*\_\_init\_\_*(pos, surf, groups)  
  
 *# tree attributes  
 self*.name = name  
 *self*.health = 5 *if self*.name == 'Small' *else* 8  
 *self*.alive = *True* stump\_path = f'../graphics/stumps/{"small" *if self*.name == "Small" *else* "large"}.png'  
 *self*.stump\_surf = pygame.image.load(stump\_path).convert\_alpha()  
  
 *# apples  
 self*.apple\_surf = pygame.image.load('../graphics/fruit/apple.png').convert\_alpha()  
 *self*.apple\_pos = APPLE\_POS[*self*.name]  
 *self*.apple\_sprites = pygame.sprite.Group()  
 *self*.create\_fruit()  
  
 *self*.player\_add = player\_add  
  
 *# sound  
 self*.axe\_sound = pygame.mixer.Sound('../audio/axe.mp3')  
 *self*.axe\_sound.set\_volume(SOUND\_VOLUME['Axe'])  
  
 *def* damage(*self*):  
 *# damaging the tree  
 self*.health -= 1  
  
 *# play sound  
 if self*.alive:  
 *self*.axe\_sound.play()  
  
 *# remove apple  
 if* len(*self*.apple\_sprites.sprites()) > 0:  
 random\_apple = choice(*self*.apple\_sprites.sprites())  
 Particle(  
 pos=random\_apple.rect.topleft,  
 surf=random\_apple.image,  
 groups=*self*.groups()[0],  
 z=LAYERS['fruit'])  
 *self*.player\_add('apple')  
 random\_apple.kill()  
  
 *def* check\_death(*self*):  
 *if self*.health <= 0:  
 Particle(*self*.rect.topleft, *self*.image, *self*.groups()[0], LAYERS['fruit'], 300)  
 *self*.image = *self*.stump\_surf  
 *self*.rect = *self*.image.get\_rect(midbottom=*self*.rect.midbottom)  
 *self*.hitbox = *self*.image.get\_rect(midbottom=(*self*.rect.midbottom[0], *self*.rect.midbottom[1]-26)).inflate(-20, -*self*.rect.height \* 0.99)  
 *self*.alive = *False  
 self*.player\_add('wood', 5 *if self*.name == 'Small' *else* 8)  
  
 *def* create\_fruit(*self*):  
 *for* pos *in self*.apple\_pos:  
 *if* randint(0, 10) < 2:  
 x = pos[0] + *self*.rect.left  
 y = pos[1] + *self*.rect.top  
 Generic(  
 pos=(x, y),  
 surf=*self*.apple\_surf,  
 groups=[*self*.apple\_sprites, *self*.groups()[0]],  
 z=LAYERS['fruit'])  
  
 *def* update(*self*, dt):  
 *if self*.alive:  
 *self*.check\_death()  
  
 *# Change volume  
 self*.axe\_sound.set\_volume(SOUND\_VOLUME['Axe'])

**esc\_menu.py**

*import* sys  
*import* pygame  
*from* settings *import* \*  
*from* timer *import* Timer  
  
  
*class* EscMenu:  
 *def \_\_init\_\_*(*self*, player, toggle):  
 *self*.player = player  
 *self*.toggle\_esc\_menu = toggle  
 *self*.display\_surface = pygame.display.get\_surface()  
 *self*.font = pygame.font.Font('../font/Pixeltype.ttf', 80)  
  
 *# import  
 self*.sound\_on\_surf = pygame.image.load(f'../graphics/overlay/sound\_on.png').convert\_alpha()  
 *self*.sound\_off\_surf = pygame.image.load(f'../graphics/overlay/sound\_off.png').convert\_alpha()  
 *self*.slot\_surf = pygame.image.load(f'../graphics/overlay/slot\_menu.png').convert\_alpha()  
 *self*.select\_slot\_surf = pygame.image.load(f'../graphics/overlay/select\_slot\_menu.png').convert\_alpha()  
  
 *# self.background\_surf = pygame.image.load(f'../graphics/overlay/setting\_clear.png').convert\_alpha()  
 # self.background\_rect = self.background\_surf.get\_rect(center=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2))  
  
 self*.background\_setting\_surf = pygame.image.load(f'../graphics/overlay/setting.png').convert\_alpha()  
 *self*.background\_setting\_rect = *self*.background\_setting\_surf.get\_rect(center=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2))  
  
 *self*.select\_setting\_surf = [*self*.background\_setting\_surf, *self*.background\_setting\_rect]  
  
 *# options  
 self*.width = 400  
 *self*.space = 10  
 *self*.padding = 8  
  
 *# movement  
 self*.index = 0  
 *self*.timer = Timer(250)  
  
 *# entries  
 self*.all\_options = ALL\_OPTIONS  
 *self*.current\_option = 'options'  
 *self*.setup(*self*.current\_option)  
  
 *# sound  
 self*.switch = pygame.mixer.Sound('../audio/switch.mp3')  
 *self*.switch.set\_volume(0.05)  
  
 *def* draw\_menu(*self*, option):  
 *# count slot* cnt = len(*self*.all\_options[option])  
  
 *try*:  
 current\_item = *self*.all\_options[option][*self*.index]  
 *except* IndexError:  
 *self*.index = 0  
 current\_item = *self*.all\_options[option][*self*.index]  
  
 *for* i *in* range(cnt):  
 *# draw slots* slot\_rect = *self*.slot\_surf.get\_rect(center=(SCREEN\_WIDTH // 2, 290 + (i \* 130)))  
 *self*.display\_surface.blit(*self*.slot\_surf, slot\_rect)  
  
 *# draw text  
 if* option == 'volume':  
 text\_surf = *self*.font.render(f'{*self*.all\_options[option][i]}', *True*, '#b68962')  
 text\_rect = text\_surf.get\_rect(topleft=(SCREEN\_WIDTH // 2 - 197, 265 + (i \* 130)))  
 *self*.display\_surface.blit(text\_surf, (text\_rect[0], text\_rect[1] + 4))  
 *else*:  
 text\_surf = *self*.font.render(f'{*self*.all\_options[option][i]}', *True*, '#b68962')  
 text\_rect = text\_surf.get\_rect(center=(SCREEN\_WIDTH // 2, 290 + (i \* 130)))  
 *self*.display\_surface.blit(text\_surf, (text\_rect[0], text\_rect[1] + 4))  
  
 *# draw selection* select\_slot\_rect = *self*.select\_slot\_surf.get\_rect(center=(SCREEN\_WIDTH // 2, 290 + (*self*.all\_options[option].index(current\_item) \* 130)))  
 *self*.display\_surface.blit(*self*.select\_slot\_surf, select\_slot\_rect)  
  
 *if* option == 'volume':  
  
 count\_on = int(SOUND\_VOLUME[current\_item] \* 10)  
 count\_off = 10  
  
 *for* i *in* range(count\_off):  
 sound\_off\_rect = *self*.sound\_off\_surf.get\_rect(center=(SCREEN\_WIDTH // 2 + 38 + (i \* 17), 286 + (*self*.index \* 130)))  
 *self*.display\_surface.blit(*self*.sound\_off\_surf, sound\_off\_rect)  
  
 *for* i *in* range(count\_on):  
 sound\_on\_rect = *self*.sound\_on\_surf.get\_rect(center=(SCREEN\_WIDTH // 2 + 38 + (i \* 17), 286 + (*self*.index \* 130)))  
 *self*.display\_surface.blit(*self*.sound\_on\_surf, sound\_on\_rect)  
  
 *def* setup(*self*, option):  
 *# create text surfaces  
 self*.text\_surfs = []  
 *self*.total\_height = 0  
 *for* item *in self*.all\_options[option]:  
 text\_surf = *self*.font.render(item, *True*, 'Black')  
 *self*.text\_surfs.append(text\_surf)  
 *self*.total\_height += text\_surf.get\_height() + (*self*.padding \* 2)  
  
 *self*.total\_height += (len(*self*.text\_surfs) - 1) \* *self*.space  
 *self*.menu\_top = SCREEN\_HEIGHT // 2 - *self*.total\_height // 2  
 *self*.main\_rect = pygame.Rect(SCREEN\_WIDTH // 2 - *self*.width // 2, *self*.menu\_top, *self*.width, *self*.total\_height)  
  
 *# volume level  
 if self*.current\_option == 'volume':  
 current\_item = *self*.all\_options[*self*.current\_option][*self*.index]  
 *if* len(str(SOUND\_VOLUME[current\_item])) == 4:  
 *if* str(SOUND\_VOLUME[current\_item])[0:2] == '1.':  
 volume\_lev = str(SOUND\_VOLUME[current\_item])[0] + str(SOUND\_VOLUME[current\_item])[2]  
 *else*:  
 volume\_lev = (str(SOUND\_VOLUME[current\_item]))[-1]  
 *if* len(str(SOUND\_VOLUME[current\_item])) == 1:  
 *if* str(SOUND\_VOLUME[current\_item]) == '1':  
 volume\_lev = str(SOUND\_VOLUME[current\_item]) + '0'  
 *else*:  
 volume\_lev = str(SOUND\_VOLUME[current\_item])  
 *if* len(str(SOUND\_VOLUME[current\_item])) == 3:  
 *if* str(SOUND\_VOLUME[current\_item])[0:2] == '1.':  
 volume\_lev = str(SOUND\_VOLUME[current\_item])[0] + str(SOUND\_VOLUME[current\_item])[2]  
 *else*:  
 volume\_lev = (str(SOUND\_VOLUME[current\_item]))[-1]  
 *self*.level\_surf = *self*.font.render(volume\_lev, *True*, 'Black')  
  
 *def* input(*self*):  
 keys = pygame.key.get\_pressed()  
 *self*.timer.update()  
  
 *if not self*.timer.active:  
 *if* keys[pygame.K\_w]:  
 *self*.index -= 1  
 *self*.switch.play()  
 *self*.timer.activate()  
  
 *if* keys[pygame.K\_s]:  
 *self*.index += 1  
 *self*.switch.play()  
 *self*.timer.activate()  
  
 *if* keys[pygame.K\_RETURN]:  
 *self*.timer.activate()  
 *self*.switch.play()  
  
 *# get item* current\_item = *self*.all\_options[*self*.current\_option][*self*.index]  
  
 *# options  
 if* current\_item == 'Play':  
 *self*.toggle\_esc\_menu()  
 *self*.timer.activate()  
  
 *if* current\_item == 'Exit':  
 pygame.quit()  
 sys.exit()  
  
 *# In options  
 if* current\_item == 'Options':  
 *self*.current\_option = 'in\_options'  
 *self*.select\_setting\_surf = [*self*.background\_setting\_surf, *self*.background\_setting\_rect]  
  
 *if* current\_item == 'Volume':  
 *self*.current\_option = 'volume'  
  
 *if* keys[pygame.K\_a]:  
 *self*.timer.activate()  
 current\_item = *self*.all\_options[*self*.current\_option][*self*.index]  
  
 *if self*.current\_option == 'volume':  
 *self*.switch.play()  
 *if* current\_item == 'Tools':  
 SOUND\_VOLUME['Tools'] -= 0.1  
 SOUND\_VOLUME['Wave'] -= 0.1  
 SOUND\_VOLUME['Axe'] -= 0.1  
 SOUND\_VOLUME['Water'] -= 0.1  
 SOUND\_VOLUME['Hoe'] -= 0.1  
 SOUND\_VOLUME['Plant'] -= 0.1  
  
 *elif* current\_item == 'Affects':  
 SOUND\_VOLUME['Affects'] -= 0.1  
 SOUND\_VOLUME['Success'] -= 0.1  
 SOUND\_VOLUME['Switch tool'] -= 0.1  
 SOUND\_VOLUME['Bye or Sell'] -= 0.1  
 *else*:  
 SOUND\_VOLUME[current\_item] -= 0.1  
  
 *if* len(str(SOUND\_VOLUME[current\_item])) > 4:  
 SOUND\_VOLUME[current\_item] = float(str(SOUND\_VOLUME[current\_item])[:4])  
 *if* str(SOUND\_VOLUME[current\_item])[-1] != '0' *and* str(SOUND\_VOLUME[current\_item])[-1] != '5':  
 SOUND\_VOLUME[current\_item] = round(float(str(SOUND\_VOLUME[current\_item])[:4]), 1)  
  
 *if* SOUND\_VOLUME[current\_item] <= 0:  
 *if* current\_item == 'Tools':  
 SOUND\_VOLUME['Tools'] = SOUND\_VOLUME['Axe'] = SOUND\_VOLUME['Water'] = SOUND\_VOLUME[  
 'Hoe'] = SOUND\_VOLUME['Plant'] = 0  
  
 *elif* current\_item == 'Affects':  
 SOUND\_VOLUME['Affects'] = SOUND\_VOLUME['Success'] = SOUND\_VOLUME['Switch tool'] = \  
 SOUND\_VOLUME['Bye or Sell'] = 0  
 *else*:  
 SOUND\_VOLUME[current\_item] = 0  
  
 *if* keys[pygame.K\_d]:  
 *self*.timer.activate()  
 current\_item = *self*.all\_options[*self*.current\_option][*self*.index]  
  
 *if self*.current\_option == 'volume':  
 *self*.switch.play()  
 *if* current\_item == 'Tools':  
 *if* SOUND\_VOLUME['Tools'] == 0:  
 SOUND\_VOLUME['Tools'] += 0.1  
 SOUND\_VOLUME['Wave'] += 0.1  
 SOUND\_VOLUME['Axe'] += 0.3  
 SOUND\_VOLUME['Water'] += 0.1  
 SOUND\_VOLUME['Hoe'] += 0.4  
 SOUND\_VOLUME['Plant'] += 0.5  
  
 *else*:  
 SOUND\_VOLUME['Tools'] += 0.1  
 SOUND\_VOLUME['Wave'] -= 0.1  
 SOUND\_VOLUME['Axe'] += 0.1  
 SOUND\_VOLUME['Water'] += 0.1  
 SOUND\_VOLUME['Hoe'] += 0.1  
 SOUND\_VOLUME['Plant'] += 0.1  
  
 *elif* current\_item == 'Affects':  
 *if* SOUND\_VOLUME['Tools'] == 0:  
 SOUND\_VOLUME['Affects'] += 0.1  
 SOUND\_VOLUME['Success'] += 0.1  
 SOUND\_VOLUME['Switch tool'] += 0.2  
 SOUND\_VOLUME['Bye or Sell'] += 0.1  
  
 *else*:  
 SOUND\_VOLUME['Affects'] += 0.1  
 SOUND\_VOLUME['Success'] += 0.1  
 SOUND\_VOLUME['Switch tool'] += 0.1  
 SOUND\_VOLUME['Bye or Sell'] += 0.1  
  
 *else*:  
 SOUND\_VOLUME[current\_item] += 0.1  
  
 *if* len(str(SOUND\_VOLUME[current\_item])) > 4:  
 SOUND\_VOLUME[current\_item] = float(str(SOUND\_VOLUME[current\_item])[:4])  
 *if* str(SOUND\_VOLUME[current\_item])[-1] != '0' *and* str(SOUND\_VOLUME[current\_item])[-1] != '5':  
 SOUND\_VOLUME[current\_item] = round(float(str(SOUND\_VOLUME[current\_item])[:4]), 1)  
  
 *if* SOUND\_VOLUME[current\_item] >= 1:  
 *if* current\_item == 'Tools':  
 SOUND\_VOLUME['Tools'] = SOUND\_VOLUME['Water'] = SOUND\_VOLUME['Wave'] = 1  
 SOUND\_VOLUME['Axe'] = 1.2  
 SOUND\_VOLUME['Hoe'] = 1.3  
 SOUND\_VOLUME['Plant'] = 1.5  
 *elif* current\_item == 'Affects':  
 SOUND\_VOLUME['Affects'] = SOUND\_VOLUME['Success'] = SOUND\_VOLUME['Bye or Sell'] = 0  
 SOUND\_VOLUME['Switch tool'] += 0.2  
 *else*:  
 SOUND\_VOLUME[current\_item] = 1  
  
 *if* keys[pygame.K\_ESCAPE]:  
 *if self*.current\_option != 'options':  
 *if self*.current\_option == 'volume' *or self*.current\_option == 'hotkeys':  
 *self*.current\_option = 'in\_options'  
 *self*.timer.activate()  
 *else*:  
 *# self.select\_setting\_surf = [self.background\_surf, self.background\_rect]  
 self*.current\_option = 'options'  
 *self*.timer.activate()  
  
 *# clamp the values  
 if self*.index < 0:  
 *self*.index = len(*self*.text\_surfs) - 1  
 *if self*.index > len(*self*.text\_surfs) - 1:  
 *self*.index = 0  
  
 *def* update(*self*):  
 *self*.display\_surface.blit(*self*.select\_setting\_surf[0], *self*.select\_setting\_surf[1])  
 *self*.input()  
 *self*.draw\_menu(*self*.current\_option)  
 *self*.setup(*self*.current\_option)