A project, using pygame tools for Yandex.Lyceum

Here we create a game, which idea is as follows - Exists a main hero, miner, which task is to explode enemy sniper towers. He is walking across the map, plants bombs under towers and explodes them. During the process of completing his goal he tries to avoid mines, those were planted by enemies to protect towers. Their location is not available in the beginning, but person can find them out with help of special tools on the map or by remembering their location when is getting exploded.

The following documentation is written in Russian language.

Игра реализована в трёх скриптах – EntryKit (он же main в этом проекте), GameProcess и generate\_level. В первом хранятся написанные на PyQt5 классы, описывающие окно авторизации / главное окно игры / окно статистики и проч.

GameProcess содержит основную реализацию игрового процесса – в частности, классы игрока и анимации процессов, связанных с минами и бомбой (BombAnimationPack), различные функции, связанные с игровым процессом

generate\_level ответственен за уровень – загружает его, отображает на экране. Содержит классы Tile (ячейка карты) и Camera.

Подробнее об устройстве скриптов смотреть ниже.

Устройство файловой системы проекта:

Все скрипты хранятся в самом каталоге проекта (не во вложенных папках). В основной папке два подкаталога: developer (в котором хранится скрипт для создания уровней – LevelMaker, документация по нему находится в файле LevelMakerDoc.txt) и data (собственно, все данные игры). Data содержит следующие подкаталоги:

~ dialogs (текстовые файлы с «диалогами» - разговорами между ГГ и комбатом)  
~ fonts (шрифты для игры)  
~ images (все изображения, используемые в игре)  
~ levels (уровни в виде текстовых файлов – их устройство описано в документации к LevelMaker’у)  
~ progress   
И файл game\_instruction, к которому обращается класс InstructionWindow из файла EntryKit для демонстрации инструкции игроку)

Папка progress содержит информацию о прогрессе пользователей в игре – их уровень, бонусы, количество уничтоженных вышек и прочее. В каталоге содержатся подкаталоги с именами пользователей. В каждом подкаталоге хранятся 2 txt файла – info.txt (с информацией об игроке – где он на карте располагается. Какие бонусы имеет и проч. в формате json) и map.txt (карта пользователя на момент последнего сохранения).

Описание файлов с кодом:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Файл “EntryKit”

В файле реализация 5-ти классов и классический шаблон запуска PyQt5-приложения. Во всех существует метод initUi (инициализация окна – связь кнопок с действиями, фоновое изображение и проч.) Классы таковы:

HelloWindow

Окно авторизации. Содержит виджеты для входа, а также регистрации. Содержит единственное поле – authorization, принимающее True при успешной авторизации. Методы:

find\_user – находит пользователя в БД в случае нажатия на кнопку «войти» либо Enter’a на клавиатуре. По завершении меняет self.authorization на True либо высвечивает сообщение об ошибке для пользователя

add\_user – открывает форму регистрации (создаёт объект класса AddUserWindow)

closeEvent – открывает главное окно пользователя (класс MainWindow) если авторизация прошла успешно.

MainWindow

Главное окно игрока. В нём присутствуют кнопки для загрузки игры, начала новой, просмотра статистики и инструкции. Содержит поля:

username – имя пользователя для корректной загрузки уровня, сохранения прогресса.

game\_continues – поле, содержащее boolean (открыто ли игровое окно). Если game\_continues == True, то невозможно запустить второе окно игры.

screen\_type – тип экрана. Необходимо для игрового процесса - периодически на игровом окне изображение карты будет сменятся информационными баннерами. Так вот тип отображаемого экрана хранится здесь. Может принимать значения [“game”, “dialog”, “buckler”, “detector”, “checkpoint”, “popup\_win”].

~ game означает основной игровой экран (поле, персонаж, вышки и прочее)

~ dialog это «диалоговое окно» - экран, на котором текстом отображаются диалоги между персонажем игрока и его руководителем (Комбатом)

~ buckler – баннер о получении щита

~ detector – баннер о получении металлоискателя

~ checkpoint – баннер о прохождении точки сохранения

~ popup\_win – “pop-up window”, всплывающее окно. Значение для вызова общей для всех информационных баннеров обработки окна.

dialog\_is\_going – boolean, идёт ли диалог на экране. Нужен для реализации стартового диалога игры (при начале новой игры)

Методы:

show\_instruction – запускает окно инструкции (класс InstructionWindow)

start\_game – начинает новую игру

load\_game – загружает начатую игру с последнего чекпоинта

reset\_userdata – обнуляет прогресс игрока (ставит на начало первого уровня, отнимает бонусы).

InstructionWindow

Окно с отображением инструкции. Методы:

init\_child\_widget – инициализация QScrollArea, содержащей саму инструкцию

return\_parent – закрытие окна (изначально с открытием «родителя» - главного окна игры, но от закрытия главного окна пришлось отказаться – на нём завязано существование окна игрового, т.е. закрытие главного окна означает закрытие игры вообще).

AddUserWindow

Окно регистрации. Содержит единственное поле parent – виджет-«родитель», то окно, к которому вернётся программа при нажатии на кнопку «назад». Методы:

add\_user – проверяет корректность введённых данных и открывает окно подтверждения, если всё в порядке

return\_parent – закрывает окно и возвращается к окну авторизации

keyPressEvent – обрабатывает переключение между строками формы с помощью стрелок (и ввод значений клавишей Enter)

ConfirmWindow

Окно подтверждения регистрации пользователя. При получении подтверждения, обрабатывает добавление пользователя в БД. Содержит единственное поле – parent (в котором содержится виджет-окно регистрации пользователя)

Методы:

add\_user – добавление пользователя с данными в БД или вывод сообщения об ошибке, если такой пользователь уже зарегистрирован.

add\_progress\_info – добавление записей о пользователе в каталог progress (подробнее об устройстве файловой системы проекта смотреть выше)

Файл GameProcess

Файл содержит всю реализацию игрового окна программы. Содержит 4 класса и 11 функций (включая game\_process\_main) Классы:

Player

Класс игрока. При инициализации принимает аргументы:

~ username – имя пользователя, зашедшего в игру  
~ binded\_screen – объект типа pygame.Surface, экран, на котором отображается игрок  
~ player\_group – pygame.sprite.Group: группа, содержащая игрока  
~ all\_sprites\_group - pygame.sprite.Group: группа, содержащая все используемые в игре спрайты  
~ tiles\_all\_group – список, состоящий из all\_sprites\_group и tiles\_group(pygame.sprite.Group, содержащий все ячейки поля)

Содержит поля:  
~ screen – «привязанный» к игроку экран  
~ username – имя пользователя-игрока  
~ bomb\_animation\_pack – объект класса BombAnimationPack, задача которого – анимировать процессы, связанные с бомбами/минами  
~ clock – pygame.time.Clock, часы, для отсчёта времени, которе персонаж должен провести на месте до возврата на точку респавна  
~ death\_timer – значение, отвечающее за кол-во времени, пройденного со смерти (нужно, чтобы не сразу возрождать персонажа)  
~ died – Boolean, показывает, мёртв ли персонаж  
~ current\_orientation – показывает, какой стороной на нас смотрит персонаж (0 = персонаж повёрнут лицом, 1 = левым боком, 2 = правым боком, 3 = спиной)  
~ last\_animation\_step – числовое значение, хранящее кадр анимации движения в ту или иную сторону. Принимает значения от 0 до 3 в процессе работы программы  
~ can\_move – Boolean, в котором отображается, разрешено ли персонажу сейчас двигаться (может, он ставит бомбу – тогда нельзя)  
~ player\_is\_moving – Boolean, двигается ли в данный момент игрок  
~ image\_width/image\_height – константы, отображающие ширину и высоту изображения главного героя  
~ full\_image – полное изображение персонажа (все кадры движения во все стороны)  
~ image – изображение персонажа в данный момент  
~ rect – наследуемый от Sprite аттрибут, задающий положение спрайта на поле  
~ detonate\_zone/attack\_zone – константы, задающие величину (в пикселях) зон детонации и поражения установленной бомбы, соответственно  
~ bomb\_planted – Boolean, демонстрирующий, установлена ли бомба  
~ bomb\_pos – позиция бомбы в пикселях от левого верхнего угла карты или список из None, если бомба не установлена  
~ destroyed\_towers – число уничтоженных игроком вышек на данном уровне  
~ level\_num – номер проходимого сейчас уровня  
~ has\_buckler – логическое значение, есть ли у игрока щит  
~ has\_detector – Boolean, есть ли у игрока метало детектор  
~ current\_checkpoint – список из двух элементов, где первый – номер чекпоинта, а второй – список из его координат, считая от левого верхнего угла поля  
~ detonated\_mines – список координат детонированных игроком мин (координат ячеек на поле, в которых содержались мины), от которых он, впрочем, не погиб (уничтожил их со щитом)  
~ map\_x\_pos/map\_y\_pos – координаты (в пикселях) центра изображения игрока, считая от левого верхнего угла карты  
~ current\_level – список, содержащий карту, по которой в данный момент бродит ГГ  
~ towers – словарь, ключи которого – координаты расположенных на карте вышек, а значения – объекты класса Tile (ячейки поля с вышками)

Методы:

~ init\_default – метод, инициализирующий некоторые поля Player’a (важен, когда нужно сбросить значения до дефолтных)   
~ move(moving\_vector) – движение игрока по карте по оси Ox на moving\_vector[0], на moving\_vector[1] по оси Oy  
~ plant\_bomb – запускает процесс установки бомбы  
~ detonate\_bomb – подрывает бомбу, а также выполняет процессы, с подрывом сопряжённые (изменяет коды задетых вышек, умертвляет игрока, если тот попал в зону поражения)  
~ change\_picture – изменяет кадр игрока в анимации движения.  
~ check\_position(main\_window) – проверяет, на какую клетку наступил игрок, и, если надо, выдаёт ему бонус или умертвляет. Main\_window (объект класса MainWindow из файла EntryKit) передаётся для изменения экрана на информационный баннер, если потребуется, посредством изменения main\_window.screen\_type.  
~ death\_protocol – комплекс действий, связанных со смертью главного героя (запуск анимации взрыва бомбы, увеличение счётчика смертей и проч.)  
~ mine\_explosion – запуск анимации взрыва с центром в точке, где находятся ноги персонажа)  
~ detect\_mine(screen) – ищет и подсвечивает мины вокруг игрока. screen – текущий экран.  
~ revival() – возрождение игрока при смерти.   
~ exploded\_cells – возвращает координаты ячеек, задеваемых установленной бомбой. НЕ ВЫЗЫВАТЬ, ЕСЛИ ДОПОДЛИННО НЕ ИЗВЕСТНО, УСТАНОВЛЕНА ЛИ БОМБА  
~ finished – возвращает boolean, проверяет, не закончил ли игрок уровень.  
~ save\_progress – сохраняет прогресс игрока (вызывается по достижении чекпоинта)

BombAnimationPack

Класс, соединяющий в себе функции, связанные с анимациями взрыва / отрисовкой границ зон поражения/детонации и проч. При инициализации принимает аргументы:

~ player (игрок, к которому привязан объект класса)  
~ time\_period (время закладки бомбы в секундах)  
~ all\_sprite\_group – pygame.sprite.Group(), содержащая все спрайты игры

Поля:

~ time\_gone – количество времени, прошедшего с момента начала закладки бомбы  
~ clock – объект класса Pygame.time.Clock  
~ time\_period – время в секундах, которое нужно затратить на установку бомбы  
~ process\_dead – boolean, демонстрирующий, закончился ли процесс установки бомбы  
~ alpha\_screen – pygame.Surface, на которой рисуются границы зон поражения/детонации бомбы. Обладает прозрачностью.  
~ x\_indent/y\_indent – значения, отображающие отступ от левого верхнего угла карты точки, в которой заложена бомба (либо точки в ногах игрока, если бомба не заложена)  
~ player – игрок, с которым связан объект класса  
~ explosion\_image – полное изображение взрыва (все кадры анимации)  
~ exp\_image\_group – pygame.sprite.Group, содержащая explosion\_image  
~ current\_exp\_image – кадр анимации взрыва на данный момент  
~ exp\_col\_cnt/exp\_row\_cnt – значения, отображающие, какой кадр необзодимо вырезать в данный момент из explosion\_image для анимации взрыва  
  
Методы:

~ update – метод, объединяющий выполнение всех анимаций  
~ update\_progress\_bar – обновление полоски процесса (отображающей, сколько осталось до закладки бомбы)  
~ update\_bomb\_borders – отрисовка зоны поражения/детонация бомбы  
~ animate\_explosion – анимация подрыва  
~ kill\_process\_bar – функция, завершающая процесс установки бомбы

DialogWindow

Окно, отображающее диалоги между ГГ и его руководителем; в частности, отображающее стартовый и финальный экраны игры. При инициализации принимает аргументы:

~ binded\_screen – экран, на котором будет отображаться диалог   
~ dialog\_name – название диалога, который будет отображён.

Поля:

~ text – весь текст диалога  
~ screen – pygame.Surface, на которой отображается диалог  
~ text\_len – длина последней строки текста   
~ last\_char\_num – номер последнего добавленного символа  
~ bg\_image – фоновое изображение  
~ current\_text – отображаемый на данный момент на экране текст  
~ new\_paragraph – boolean, отражающий необходимость перевода на новую строку  
~ font – шрифт написания  
~ add\_char\_period – время, через которое нужно добавлять новый символ, в миллисекундах  
~ add\_paragraph\_period – время, через которое нужно добавлять абзац (также в миллисекундах)  
Методы:

~ load\_dialog – инициализация поля text  
~ show\_dialog – отображение текста, на данный момент хранящегося в current\_text  
~ change\_text – изменение current\_text в случае необходимости  
~ write\_text – рендер текста на self.screen  
~ make\_text\_array – формирование списка из строк текста

LeaveGameWindow

Класс, наследуемый от QWidget. Является «виджетом-подтверждением» (всплывает, когда пользователь собирается закрыть окно игры) с вопросом «Вы уверены, что хотите выйти»? Сохраняет информацию о количестве смертей игрока в info.txt в случае положительного ответа на вопрос. Его поле running определяет, будет ли продолжаться основной цикл игры. При инициализации принимает аргумент player. Содержит поля:

~ running – boolean, продолжать ли основной цикл игры  
~ player   
 Содержит один метод: close\_game, сохраняющий информацию о смертях и закрывающий игру.

Функции файла:

~ load\_image(name) – загрузка изображения по имени name  
~ count\_player\_coords\_p(player) – находит местоположение игрока на матрице карты по игроку (аргумент player)  
~ count\_player\_coords\_c(x, y) – находит местоположение игрока на матрице карты по координатам от левого верхнего угла карты  
~ draw\_icon (screen, image, pose) – отрисовка иконки бонуса  
~ change\_level(player, tiles\_all\_group) – смена уровня – изменения данных в info.txt, map.txt; повторная инициализация объекта класса Player, повторная отрисовка уровня, и проч.)  
~ detector\_screen – информационный баннер о получении металлоискателя  
~ chekpoint\_screen – информационный баннер о прохождении точки сохранения  
~ buckler\_screen – информационный баннер о получении щита  
~ dialog\_win(dialogname, screen, main\_window) – использования DialogWindow для отображение диалога на экране  
~ game\_process\_main(username, main\_window) – главная функция GameProcess, реализующая сам игровой процесс

Файл generate\_level

Содержит 2 важные константы: tile\_images (ключи – объекты, значения - изображения) и tile\_types – (ключи – это условное обозначение объекта на карте, значения – объект, связанный с этим обозначением), а также классы Tile («плитка» - ячейка карты) и Camera; функции load\_image, load\_level, generate\_level.

Классы:

Tile

Наследуемый от pygame.sprite.Sprite класс, описывающий ячейку поля (в какой раз это прописывается, третий?)). Не содержит методов. При инициализации принимает аргументы:

~ pos\_x/pos\_y (позиция ячейки на карте – столбец ячеек и строка соответственно)  
~ tile\_image (изображение, соответствующее данному типу ячейки)  
~ tiles\_all\_group (объект класса pygame.sprote.Group, группа всех ячеек карты)

~ tile\_type – тип ячейки (мина/вышка/пустая и тп)

Поля:

image – изображение, соответствующее ячейку

rect – описание местоположения на экране

type – тип ячейки

Camera

Поля:

~ dx – сдвиг камеры по оси Ox  
~ dy – сдвиг по Oy

Методы:

~ update(player) – корректирует сдвиг камеры, в зависимости от пришедшей информации о движении игрока, а также изменяет отступы x\_indent/y\_indent класса BombAnimationPack, дабы корректно отрисовывать границы поражения/детонации  
~ apply(all\_sprites) – двигает объекты. Аргумент – pygame.sprite.Group() со всеми объектами поля.

Функции:

~ load\_level(username) - считывает карту пользователя {username} и возвращает двумерный список, её содержащий

~ generate\_level(level, tiles\_all\_group) – отрисовка уровня