

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Забайкальский государственный университет»
(ФГБОУ ВО «ЗабГУ»)
Энергетический Факультет
Кафедра информатики вычислительной техники, прикладной математики


КУРСОВАЯ РАБОТА

по программированию

на тему «Компьютерная игра “BomberMan ”»

Выполнил студент группы ИВТ-17-1 Демидов Владимир Юрьевич

Руководитель работы: старший преподаватель, Ветров Сергей Владимирович

Отлично



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Забайкальский государственный университет»
(ФГБОУ ВО «ЗабГУ»)
Энергетический факультет
Кафедра информатики, вычислительной техники и прикладной математики

ЗАДАНИЕ
на курсовую работу

По дисциплине Программирование
Студенту Демидову Владимиру Юрьевичу
специальности: Информатика и вычислительная техника

1. Тема курсовой работы: Компьютерная игра "BomberMan "
2. Срок подачи студентом законченной работы: 19 сентября 2018 г.
3. Исходные данные к работе: источники в сети интернет
4. Перечень подлежащих разработке в курсовой работе вопросов:
 - а) Техническое задание
 - б) Создание игры
5. Перечень графического материала (если имеется): -

Дата выдачи задания «12» февраля 2018г.

Руководитель курсовой работы (проекта)  /Ветров Сергей Владимирович/

Задание принял к исполнению

«12» ФЕВРАЛЯ 2018г

Подпись студента  /Демидов Владимир Юрьевич /

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Забайкальский государственный университет»
(ФГБОУ ВО «ЗабГУ»)
Энергетический факультет
Кафедра информатики, вычислительной техники и прикладной математики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по программированию

на тему «Компьютерная игра “BomberMan”»

Выполнил студент группы ИВТ-17-1 Демидов Владимир Юрьевич

Руководитель работы: старший преподаватель, Ветров Сергей Владимирович

РЕФЕРАТ

Пояснительная записка – 37 страниц, 6 – рисунков, 0 – таблиц, 0 – схем, 0 – диаграмм, 1 – приложение.

ФУНКЦИИ, МОДУЛИ, АЛГОРИТМ, ИГРА, БИБЛИОТЕКА, ТЕКСТОВЫЙ ФАЙЛ, КЛАСС, ПОЛЬЗОВАТЕЛЬ, УРОВЕНЬ, PYTHON, PYGAME.

В работе описывается создание компьютерной игры на высокоуровневом языке Python с помощью графической библиотеки Pygame. Данная библиотека создана для упрощенной разработки игр. В работе рассматриваются модульная декомпозиция, алгоритм инициализации игры, алгоритм отрисовки каждого кадра, функции, осуществляющие взаимодействие пользователя и игры, создание и удаление игровых объектов. Рассматривается структура данных отдельного игрового объекта, загрузка приложением данных об уровнях из текстовых файлов, руководство пользователю.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	6
1. Техническое задание.....	7
2. Создание игры.....	10
2.1 Алгоритм инициализации игры.....	10
2.2 Диаграмма модулей.....	12
2.3 Распределение сущностей на классы.....	14
2.4 Покадровая отрисовка событий.....	15
2.5 Классы сущностей.....	16
2.6 Смена карты.....	20
2.7 Главный цикл в модуле main.....	21
3. Руководство пользователя.....	23
3.1 Игровой процесс.....	23
3.2 Поражение.....	27
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	28
ПРИЛОЖЕНИЕ.....	29

ВВЕДЕНИЕ

В наше время компьютерные игры очень популярны. Когда-то игры были доступны только на приставках, затем они появились и на компьютерах, а сейчас игры есть и на телефонах, и на отдельных портативных игровых девайсах. Игры очень увлекательны – они дают возможность почувствовать себя кем-то другим, прожить чужую жизнь. Но, несмотря на популярность игр, почти никто не знает о том, как на самом деле они работают.

Для написания игры использовался высокоуровневый язык программирования Python. На рассмотрение была взята графическая библиотека Pygame.

Pygame - набор модулей (библиотек), предназначенный для написания преимущественно двухмерных компьютерных игр и мультимедиа-приложений. Этот пакет предоставляет разработчику все необходимые инструменты для разработки и создания своего приложения.

1. Техническое Задание

Игра «BomberMan»

Назначение и область применения:

“BomberMan” это компьютерная игра в жанре Лабиринт (Maze). Данная игра, как тема для курсовой работы, создана в целях обучения создания компьютерных программ и не несет за собой коммерческих целей. Данная игра не является частью одноименной серии игр BomberMan а является воссозданной по игре детства копией игры. Игра бесплатна и доступна в открытом доступе для пользователей интернета.

Взаимодействие игры с другими компонентами:

В игре BomberMan взаимодействие с игрой происходит при помощи клавиатуры:

W – Движение вперед

S – Движение назад

A – Движение влево

D – Движение вправо

Стрелка влево – положить бомбу

Вывод информации производится на монитор.

Функции и задачи игры, цель игры:

“BomberMan” – игра в жанре Лабиринт, характеризующийся тем, что успех игрока определяется в основном навигацией и ориентацией в лабиринте. Сама игра не является лабиринтом, но ее стиль соответствует именно этому жанру. Игра предназначена для развлечения и проведения свободного досуга.

В игре вы – Главный герой. Вы разбились на своем корабле на неизвестной планете. Ваша задача пройти несколько уровней по планете и зачистить их, чтобы починить свой корабль и вернуться в космос.

В игре будут три разных уровня-островка, на каждом свои уникальные противники, и у каждого из них внешность и способности меняются при переходе с одного уровня на другой.

У Главного героя всего три жизни, если Главный герой умирает, то нужно начать прохождение уровня, на котором умер, заново. Когда количество жизней Главного героя кончается, то продолжить игру невозможно, нужно начать сначала. Главный герой умирает при получении любого урона.

Каждый уровень – небольшой островок в виде лабиринта, отделенного от остальных островков водой, лавой или другой преградой (стеной, камнями, и.т.п.). Для перехода на следующий уровень (островок) необходимо зачистить нынешний уровень. После зачистки происходит телепортация на следующий уровень.

Характеристики пользователя и разработчика:

Игру “BomberMan” можно запустить с компьютера, ограничение по возрасту +8. Для пользователей доступна только геймплейная часть. Для разработчика доступны геймплейная часть, а также код игры с возможностью для тестирования (читерства).

Требования к составу и параметрам тех. Средств:

Процессор: Intel® Core™ i7-4770 CPU не более 3.40 Hz;

Оперативная память: не менее 4Гб;

Свободное место на HDD: не менее 100Мб;

Устройство взаимодействия с пользователем: клавиатура, мышь;

Видеоадаптер и монитор: разрешение не менее 1000x700

Требования к программной совместимости:

Интерпретатор Python 3, модуль pygame.

1. Создание игры

1.1 Алгоритм инициализации игры

- **Входные данные**

1. Загрузка данных из файла (полная генерация уровня)
2. Обработка нажатий пользователем клавиш

- **Инициализация основных переменных**

1. lvl – Номер уровня(с нуля)
2. run – Условие работы главного цикла
3. screenWidth – Ширина экрана
4. screenHeight – Высота
5. win – Главное окно
6. entityTypes – Список всех считанных сущностей
7. blocksList – Список всех блоков карты
8. enemiesList – Список всех противников карты
9. bombsPlaced – Список всех активированных бомб
10. bomberMan – Переменная Главный Герой
11. lvlNum – Номер текущего уровня (для перехода)
12. playerWalkLeft – Список спрайтов Героя (влево)
13. playerWalkRight – Список спрайтов героя (вправо)
14. playerWalkUp – Список спрайтов героя (вверх)
15. playerWalkDown – Список спрайтов героя (вниз)
16. wolfWalkLeft - Список спрайтов Волка (влево)
17. wolfWalkUp - Список спрайтов Волка (вверх)
18. wolfWalkDown - Список спрайтов Волка (вниз)
19. slimeWalkLeft - Список спрайтов Слизня (влево)
20. slimeWalkUp - Список спрайтов Слизня (вверх)
21. slimeWalkDown - Список спрайтов Слизня (вниз)
22. batWalkLeft - Список спрайтов Мыши (влево)
23. batWalkUp - Список спрайтов Мыши (вверх)

- 24. batWalkDown - Список спрайтов Мыши (вниз)
- 25. bombPlaced - Список спрайтов бомбы
- 26. bombItselfBoom – Спрайт взорвавшейся бомбы
- 27. bombWaves - Список спрайтов взрывной волны
- 28. levelMap - Список заднего уровня (фон)
- 29. l0BBlock - Список спрайтов ломаемого блока
- 30. l0UBlock - Список спрайтов неломаемого блока
- 31. bombPlaceSound – Звук брошенной бомбы
- 32. bombBoomSound – Звук взрыва
- 33. enemyDeadSounds – Список звуков смерти противников

- **Подготовка к запуску игры**

- 1. Считывание типов сущностей из файла в переменную entityTypes
- 2. Распределение сущностей по классам (блоки, противники, и.т.п.)

2.2 Диаграмма модулей

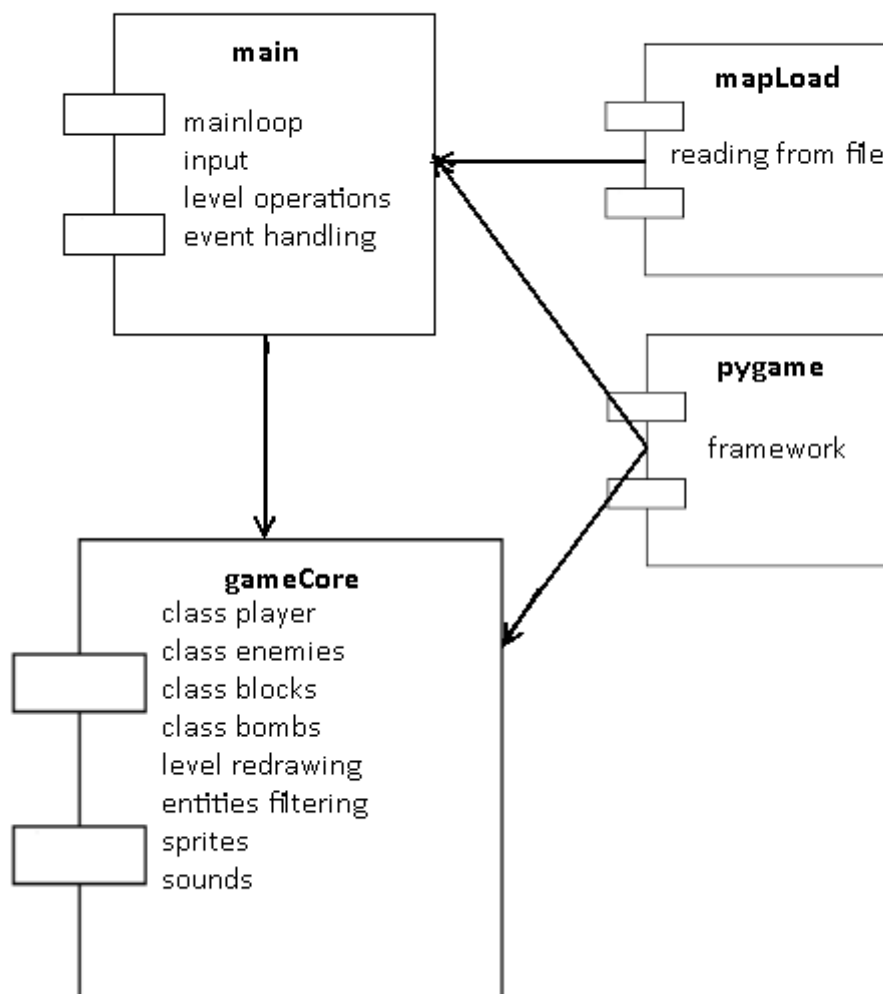


Рисунок 1 – Диаграмма модулей

Модуль **main** – главный, связывающий модуль. Отвечает за передачу в игровое ядро информации по генерации карты, передачу ввода пользователя, а также отвечает за смену уровня или закрытия игры. (см. Рис. 1).

Модуль **mapLoad** – отвечает за считывание информации по уровням из текстового файла и их передаче в главный модуль.(см. Рис. 1).

Модуль **pygame** – каркас (framework). Набор модулей (библиотек) для разработки. (см. Рис. 1).

Модуль **gameCore** – игровое ядро. В нем происходит отображение всего, что происходит, проверки на коллизии, удаление тех или

иных сущностей. Хранит в себе классы всех сущностей, а также все спрайты и звуковые эффекты. (см. Рис. 1).

2.3 Распределение сущностей на классы

- **Функция MAPLOAD модуля gameCore**

При запуске игры распределяет считанные с файла данные в списке entityTypes в соответствующие классам списки: blockList, enemiesList. При смене карты повторно вызывается для считывания другого текстового файла и повторного заполнения предварительно обнуленных списков.

Для каждой новой карты различны типы противников – функция сортирует их в список врагов с присваиванием соответствующему противнику свой набор атрибутов (для волков – скорость, а для слизней – медлительность)

Блоки распределяются по их переданному типу – есть блоки, определяющие границу за которую Главному герою заходить нельзя. Есть блоки, которые можно разрушить взрывом, – они с вероятностью 50% генерируются по всей карте.

Главный герой (BomberMan) тоже сущность и тоже имеет свой класс, но так как данная функция необходима для распределения сущностей, считанных с текстового файла, по классам, Главный герой уже имеет свой класс, и появляется он в точно заданных координатах.

Для очистки всего «мусора» (при переходе на другой уровень или смерти Главного героя) все списки с ненужной информацией необходимо удалить. Для этого предварительно вызывается функция NEWMAP в модуле gameCore.

2.4 Покадровая отрисовка событий

- **Функция `redrawGameWindow` модуля `gameCore`**

Отрисовка всех координатных изменений, а также исчезновение или появление чего-либо, осуществляется в данном модуле. Функция активируется каждый кадр (из главного модуля `main`) и отрисовывает все существующие сущности. Отрисовка происходит таким образом: все новое, что мы нарисуем, перекроет то, что было нарисовано ранее – поэтому сначала функция отображает соответствующую номеру уровня карту, затем размещает на нее все остальное: неразрушимые блоки-препятствия, разрушаемые блоки, противников, и соответственно Главного героя.

Так как есть сущности, которые в данный момент не существуют (не объявлены) но они могут появиться в любой момент (бомбы и взрывные волны от них), необходимо каждый кадр проверять, существуют ли эти сущности или нет. Если существуют, происходит их отрисовка. В конце всех изменений происходит обновление экрана.

2.5 Классы сущностей

- **Класс entity в модуле gameCore**

При распределении всех сущностей появились новые классы. Самый важный – класс блоков. Блоки – геймплейная механика, они позволяют создать лабиринт и четко задать границы, за которые нельзя заходить Главному герою.

Каждый блок состоит из координат по осям X и Y, ширины и высоты, Типом сущности (нерушимые блоки, рушимые блоки, которые будут сгенерированы с вероятностью 50%, и рушимые блоки со 100% вероятностью появления для ограничения передвижения врагов), обладает своим «хитбоксом» (квадрат который определяет физическое существование блоков, «нормальная сила», для проверок пересечений с другими хитбоксами других сущностей).

В себе класс содержит функцию покадровой самоотрисовки (в зависимости от типа блока дает ему нужный спрайт).

- **Класс bomb в модуле gameCore**

Самый сложный класс, так как бомба – основная механика прохождения игры. Само название игры содержит в себе «бомбу». Каждая бомба состоит из координат по осям X и Y, ширины, высоты, хитбокса и показатель таймер бомбы. Таймер отвечает за быстрое отображение взрыва бомбы.

В себе класс содержит несколько функций. Функция покадровой самоотрисовки, которая рисует взрыв самой бомбы (не ее взрывной волны). Функция Timer, которая, работая со счетчиком заполняет переменные таймеров для каждой бомбы (у каждой бомбы свой таймер). Функция KaBoom, которая оперирует с рядом данных и имеет сложные процессы. В ней, через каждый ее вызов, обнуляются списки с координатами взрывных волн (во все 4 стороны). Затем, происходит определение – нужно ли рисовать волны. Если встретился блок на пути – дальнейшее объявление координат волн не нужно. Затем внутри

функции kaBoom происходит вызов следующей функции waveDestruction, которая уничтожает все что попадется на ее пути. Каждую из четырех волн необходимо сравнивать с заново сгенерированным списком блоков, чтобы не уничтожить блоки под теми индексами. Также, если под волну попался противник, происходит соответствующий для типа противника звук смерти. Если главный герой попал под волну. его тоже убьет.

Последняя функция в классе bomb – функция bombWaveDraw, она рисует волны во все 4 стороны. После истечения таймера переменная «самосуществования» меняется, и отрисовки несуществующих волн (списков) не произойдет.

Для того чтобы волны уничтожали в точности 1 блок на своем пути и не шли на пересечении двух блоков и выбирали один из них вызывается функция calcPlayerSuperPos в модуле gameCore, которая вычисляет абсолютные координаты игрока (координаты верхнего левого угла клетки). Положив бомбу в эти координаты, волны всегда будут попадать точно в 1 блок перед собой. Также метод абсолютных координат позволяет исключить «читерство» - когда чтобы взорвать противника достаточно положить бомбу на пути противника, зайдя всего на 1 пиксель. Теперь чтобы бомба легла точно на пути противника. необходимо зайти хотя бы на 16 пикселей (половина пути).

- **Класс player в модуле gameCore**

В классе объявлены оставшиеся жизни, показатель попадания по игроку, координаты по осям X и Y, ширина и высота, скорость, показатели направления, счетчик шагов (для корректного отображения 9 спрайтов за 27 кадров), показатель того что игрок стоит на месте, его хитбокс, и коллизии 4х сторон центра игрока (для очень удобного передвижения по лабиринту).

Класс содержит несколько модулей. Модуль покадровой самоотрисовки, которая, в зависимости от показателя шагов рисует походку игрока по направлению, которое задает пользователь. Если же игрок не двигается то он остановится и будет смотреть в том направлении, в которое шел до этого. В этой же функции происходит передвижение квадрата с определенными сторонами. Из этой функции вызывается следующая, функция collision, если хитбокс игрока пересек хитбокс противника, показатель попадания меняется и уровень генерируется заново.

- **Класс enemy в модуле gameCore**

Второй по сложности класс в модуле, поскольку противники варьируются в зависимости от уровня, и у каждого своя скорость и механика поведения. Каждый противник обладает координатами X и Y, шириной и высотой, счетчиком шагов (для корректного рисования анимации шагов), скоростью по двум осям, скоростью по умолчанию, бонусной скоростью, хитбоксом, дополнительным хитбоксом с определенными сторонами, счетчиками события и действия, типом противника, и списками спрайтов для анимации движения. В соответствии с уровнем, в класс передается свой список – на каждом уровне будет новый противник. Также, меняются и скорость по умолчанию и бонусная скорость. Таким образом волк может внезапно перейти на быстрый бег и догнать главного героя, или бегать туда-сюда, а потом выйти в любом направлении.

Класс содержит несколько модулей. Модуль покадрового рисования работает аналогично модулю рисования в классе Player модуля gameCore. Но теперь из модуля отрисовки вызываются два других – модуль запроса спрайтов getEnemySprites (для каждого уровня – свой противник) и модуль вычисления поведения противника moveLogic. В функции поведения противника весомую роль играет счетчик событий и генератор случайных значений. Благодаря им,

каждые 2-3 секунды происходит событие: либо внезапное ускорение для волка (или замедление для слизня), либо внезапный хаотичный бег кругами, и полная неопределимость направления движения волка.

Также, при столкновении с блоком противник поворачивает влево или вправо и бежит дальше до нового столкновения или до происхождения события. При пересечении хитбокса блока и одного из четырех сторон треугольника хитбокса противника его отталкивает назад.

2.6 Смена карты

- **Функции levelOperations и onPlayerHit в главном модуле main**

Изначально при запуске игры один раз вызывается функция **levelOperations**, которая, в зависимости от показателя текущего уровня (реализован в виде счетчика, при смене уровня увеличивается на 1) выбирает, что делать с картой. При первом вызове происходит добавление в счетчик уровня, затем отчистка всех списков от «мусора», считывание из текстового файла типов сущностей и передача их в список, распределение по классам сущностей, и появление Главного героя.

Условие что список врагов пуст дает возможность использовать эту часть кода не только для начала игры после запуска, но и для смены уровня при полной зачистке.

Если игрок получил урон – удаляются все списки и уровень генерируется заново.

Если игрок исчерпал количество жизней – происходит выход из игры.

Функция onPlayerHit вызывается только при получении игроком урона. Переменная урона по игроку обнуляется, вызывается функция levelOperations с передаваемым значением количества жизней, затем вычитается одна жизнь.

2.7 Главный цикл в модуле main

- **Передвижение Главного героя**

Главный цикл – сердце игры. Без него не будет работать ничего. В нем происходят самые главные процессы. Передвижение осуществляется клавишами W,A,S,D , для которых необходимо проверять нажатие одной из клавиш каждый кадр. Если происходит коллизия между блоком и стороной хитбокса игрока, которая ближе всего к блоку – отключается возможность передвижения в этом направлении, и происходит толчок в обратном направлении. Данная механика позволяет «помогать» Главному герою переходить из движения по одной оси на движение по другой, если коллизия произошла одновременно у двух сторон треугольника (угол треугольника) – происходит толчок по обеим осям, и игрок становится в позицию, в которой нет коллизий.

Для постоянной проверки коллизий необходимо постоянно обновлять хитбоксы Главного героя и всех сущностей. Для осуществления удобного перемещения в главном цикле постоянно обновляются координаты сторон игрока.

- **Заложение бомбы**

Для заложения бомбы тоже необходимо постоянно проверять нажата ли клавиша бомбы, и еще несколько факторов: количество бомб не выше допустимого, и бомбы не лежат друг на друге. После того как ставится бомба, в соответствующий список добавляется переменная класса `bomb`, и задается таймер. По истечении таймера одна из проверок события в главном цикле станет положительной, и бомба взорвется.

Три переменные таймеров для двух бомб заданы специально: иногда бывает переполнение списка бомб, если бы на разрешенные две бомбы было ровно две таймерные переменные – одна бомба никогда бы не взорвалась. Третья переменная исключает подобный исход.

- **Проверка получения игроком урона**

В цикле также необходимо постоянно проверять был ли получен урон. Если игрок получил урон, то вызываются соответствующие функции, и затем переменная обнуляется.

- **Кадры в секунду**

Ограничение кадров в секунду следует задать в самом начале цикла. Это не обязательно, но следует отображать каждый спрайт через равный промежуток времени. Это сделает анимацию более естественной.

3. Руководство пользователя

3.1 Игровой процесс

Для передвижения используются клавиши W,A,S,D. Двигаться разрешено только внутри игрового поля, которое окружено темными камнями (см. Рис. 2).



Рисунок 2 – Границы игрового поля

На пути будут встречаться камни светлого цвета. Их можно уничтожить бомбой. Положить бомбу можно клавишей Влево.



Рисунок 3 – Бомба

Главная задача – зачистить уровень. Чтобы убить противника необходимо положить бомбу, чтобы его задело взрывной волной.



Рисунок 4 – Взрывная волна

Каждый противник опасен по-своему! Волки непредсказуемы, и могут внезапно начать бежать, а бегают они в два раза быстрее игрока! Слизни же медлительны, но, если он зажмет вас в углу, то вы будете мучительно ожидать пока либо ваша бомба вас не убьет, либо слизень до вас не доползет. Его скорость обманчива, иногда он тоже может ускориться.



Рисунок 5 – Медлительный слизень

3.2 Поражение

Не только противники могут убить игрока, своя же взрывная волна тоже опасна! После установления бомбы необходимо отбежать в безопасное место! Бомба взрывается в виде Плюса.



Рисунок 6 – Безопасное место

У игрока всего три жизни. При получении урона весь прогресс на данном уровне обнуляется, и необходимо начать уровень заново. Если жизни кончились – игра закрывается.

ЗАКЛЮЧЕНИЕ

В результате выполнения данной курсовой работы выявлено, что для создания игры стоит следовать строгим правилам. При создании игры необходим главный цикл, который будет считывать данные, покадровые изменения объектов, проверять коллизии тех или иных объектов. Любое движение обусловлено кадрами. Прежде чем отрисовывать изменения необходимо внести эти изменения – поменять координаты объекта или вовсе его удалить или добавить. После изменения координат происходит очередной запрос на отрисовку из главного цикла. Для более естественной и плавной анимации передвижения сущностей стоит менять спрайт каждое определенное число кадров. Также значительно упрощает разработку метод суперпозиции. Для того чтобы строго задать место падения взрывной волны достаточно округлять координаты до верхнего левого угла квадрата. Тогда вместо двух блоков взрыв заденет всего один. Также значительно упрощает разработку использование двух переменных скоростей – по двум осям. Они позволяют задать не только строго ориентированное движение, но и хаотичные неопределимые. Даже возможно, при условии что игра не линейная, задать сущности скорость по диагонали. Дополнительно, можно добавить в игру систему бонусов для улучшения геймплея. Можно добавить и главного босса, у которого свои уникальные способности, по которому нужно несколько раз попасть бомбой. Потенциал развития игры безграничен, и с развитием способностей к программированию становится легче разрабатывать игры и решать определенные баги и проблемы, а также дорабатывать игру и добавлять в нее новый интересный контент.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Wikipedia. Bomberman [Электронный ресурс]//Электронный Текстовый документ. Режим доступа:
[https://ru.wikipedia.org/wiki/Bomberman_\(%D1%81%D0%B5%D1%80%D0%B8%D1%8F_%D0%B8%D0%B3%D1%80\)](https://ru.wikipedia.org/wiki/Bomberman_(%D1%81%D0%B5%D1%80%D0%B8%D1%8F_%D0%B8%D0%B3%D1%80)) свободный
2. Wikipedia. Жанр Лабиринт [Электронный ресурс]//Электронный Текстовый документ. Режим доступа:
[https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D0%B1%D0%B8%D1%80%D0%B8%D0%BD%D1%82_\(%D0%B6%D0%B0%D0%BD%D1%80\)](https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D0%B1%D0%B8%D1%80%D0%B8%D0%BD%D1%82_(%D0%B6%D0%B0%D0%BD%D1%80)) свободный
3. pygame. [Электронный ресурс]//Электронный Текстовый документ. Режим доступа: <https://www.pygame.org/news> свободный
4. stackoverflow. [Электронный ресурс]//Электронный Текстовый документ. Режим доступа: <https://stackoverflow.com/> свободный
5. Sprite library. [Электронный ресурс]//Электронный Текстовый документ. Режим доступа: <https://opengameart.org/content/2d-complete-characters> свободный

ПРИЛОЖЕНИЕ

Исходный код программы

Модуль main

```
import pygame
import gameCore
import mapLoad

def levelOperations(remainingLives = 0):
    global lvl
    if len(gameCore.enemiesList) == 0 and lvl < 2:
        lvl += 1
        gameCore.NEWMAP(lvl)
        gameCore.entityTypes = mapLoad.Load(lvl)
        gameCore.MAPLOAD()
        gameCore.bomberMan = gameCore.player(160, 96, 32, 32)
    if remainingLives in [2, 3, 4, 5]:
        gameCore.enemiesList[:] = []
        gameCore.NEWMAP(lvl)
        gameCore.entityTypes = mapLoad.Load(lvl)
        gameCore.MAPLOAD()
        gameCore.bomberMan = gameCore.player(160, 96, 32, 32)
    elif remainingLives == 1:
        print("GAME OVER!")
        run = False
        pygame.quit()
        quit()

def onPlayerHit(remainingLives):
    print("HIT! -1 LIFE!")
    gameCore.bomberMan.HIT = False
    levelOperations(remainingLives)
    gameCore.player.remaininglives -= 1

pygame.init()
clock = pygame.time.Clock()

lvl = -1
pygame.mixer.music.play(-1)
levelOperations()

run = True
while run:
    clock.tick(27)
    #COLLISON CHECKING
    bomberManRect = pygame.Rect(gameCore.bomberMan.hitbox)
    #PLAYER COLLISION
    bomberManBoxL = pygame.Rect(gameCore.bomberMan.boxL)
    bomberManBoxR = pygame.Rect(gameCore.bomberMan.boxR)
    bomberManBoxU = pygame.Rect(gameCore.bomberMan.boxU)
    bomberManBoxD = pygame.Rect(gameCore.bomberMan.boxD)
    #BLOCKS COLLISION
    blockRects = []
    for i in range(len(gameCore.blocksList)):
        blockRects += [pygame.Rect(gameCore.blocksList[i]\
            .hitbox)]
    collideList = bomberManRect.collidelistall(blockRects)
    #PLAYER COLLISION
    CollisionBoxL = bomberManBoxL.collidelistall(blockRects)
    CollisionBoxR = bomberManBoxR.collidelistall(blockRects)
    CollisionBoxU = bomberManBoxU.collidelistall(blockRects)
    CollisionBoxD = bomberManBoxD.collidelistall(blockRects)
    #BOMBS COLLISION
```

```

bombRects = []
for i in range(len(gameCore.bombsPlaced)):
    bombRect = pygame.Rect(gameCore.bombsPlaced[i].hitbox)
    bombRects += [bombRect]

for event in pygame.event.get():
    if event.type == pygame.QUIT:
        run = False
        pygame.quit()
        quit()
    #bomb timer events reseters
    if event.type == pygame.USEREVENT+1:
        pygame.mixer.Sound.play(gameCore.bombBoomSound)
        gameCore.bomb.kaBoom()
        pygame.time.set_timer(pygame.USEREVENT+1, 0)
    if event.type == pygame.USEREVENT+2:
        pygame.mixer.Sound.play(gameCore.bombBoomSound)
        gameCore.bomb.kaBoom()
        pygame.time.set_timer(pygame.USEREVENT+2, 0)
    if event.type == pygame.USEREVENT+3:
        pygame.mixer.Sound.play(gameCore.bombBoomSound)
        gameCore.bomb.kaBoom()
        pygame.time.set_timer(pygame.USEREVENT+3, 0)

if gameCore.bomberMan.HIT:
    onPlayerHit(gameCore.player.remaininglives)

    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT] and len(gameCore.bombsPlaced) < 2\
        and not(bomberManRect.collidelistall(bombRects)):
        pygame.mixer.Sound.play(gameCore.bombPlaceSound)
        gameCore.bomb.Timer()
        gameCore.bombsPlaced += [gameCore.bomb(32, 32)]

    if keys[pygame.K_a] and not CollisionBoxL:
        gameCore.bomberMan.x -= gameCore.bomberMan.vel
        gameCore.bomberMan.left = True
        gameCore.bomberMan.right = False
        gameCore.bomberMan.up = False
        gameCore.bomberMan.down = False
        gameCore.bomberMan.standing = False
    elif keys[pygame.K_d] and not CollisionBoxR:
        gameCore.bomberMan.x += gameCore.bomberMan.vel
        gameCore.bomberMan.right = True
        gameCore.bomberMan.left = False
        gameCore.bomberMan.up = False
        gameCore.bomberMan.down = False
        gameCore.bomberMan.standing = False
    elif keys[pygame.K_w] and not CollisionBoxU:
        gameCore.bomberMan.y -= gameCore.bomberMan.vel
        gameCore.bomberMan.up = True
        gameCore.bomberMan.down = False
        gameCore.bomberMan.left = False
        gameCore.bomberMan.right = False
        gameCore.bomberMan.standing = False
    elif keys[pygame.K_s] and not CollisionBoxD:
        gameCore.bomberMan.y += gameCore.bomberMan.vel
        gameCore.bomberMan.down = True
        gameCore.bomberMan.up = False
        gameCore.bomberMan.left = False
        gameCore.bomberMan.right = False
        gameCore.bomberMan.standing = False
    else:
        gameCore.bomberMan.standing = True
        gameCore.bomberMan.walkCount = 0

    if CollisionBoxL:
        gameCore.bomberMan.x += 7

```

```
if CollisionBoxR:
    gameCore.bomberMan.x -= 7
if CollisionBoxU:
    gameCore.bomberMan.y += 7
if CollisionBoxD:
    gameCore.bomberMan.y -= 7

gameCore.redrawGameWindow()
levelOperations()
```

Модуль mapLoad

```
def Load(lvlNum):
    Line = []
    ListofLine = []
    List = []
    file_f = open('data\levels\lvl'+str(lvlNum)+'.txt', 'r')
    for i in range(30): #we have 30 rows of entities
        Line = file_f.readline()
        ListofLine += Line
        ListofLine.pop(-1) #we remove the '\n' symbol
        List += [ListofLine]
        ListofLine = []
    file_f.close()
    return List
```

Модуль gameCore

```
import pygame
import random

class entity(object):
    def __init__(self, x, y, width, height, entityType):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.entityType = entityType
        self.hitbox = (self.x, self.y, self.width, self.height)

    def draw(self, win):
        if self.entityType == '1':
            win.blit(l0UBlock, (self.x, self.y))
        elif self.entityType in ['2', '3']:
            win.blit(l0BBlock, (self.x, self.y))

class bomb(object):
    powerLvl = 5
    def __init__(self, width, height):
        self.x = calcPlayerSuperPos('X')
        self.y = calcPlayerSuperPos('Y')
        self.width = width
        self.height = height
        self.hitbox = (self.x, self.y, self.width, self.height)
        self.bombTimer = 0

    def draw(self, win):
        if self.bombTimer + 1 >= 27:
            self.bombTimer = 0
        else:
            self.bombTimer += 1
        win.blit(bombPlaced[self.bombTimer//3], (self.x, self.y))

    Cnt = 1
    def Timer():
        pygame.time.set_timer(pygame.USEREVENT+bomb.Cnt, 3000)
    #3 seconds till boom
    bomb.Cnt+=1
    if bomb.Cnt > 3:
        bomb.Cnt = 1

    waveExists = False
    waveLeft = []
    waveRight = []
    waveUp = []
    waveDown = []
    waveItself = []
    def kaBoom():
        bomb.waveLeft = []
        bomb.waveRight = []
        bomb.waveUp = []
        bomb.waveDown = []
        bomb.waveItself = []

    tempBlockRects = []
    for j in range(len(blocksList)):
        tempBlockRects += [pygame.Rect(blocksList[j].hitbox)]
    for i in range(bomb.powerLvl):
        bomb.waveLeft += [bombsPlaced[0].x-(32*(i+1)),\
```

```

        bombsPlaced[0].y]
    if pygame.Rect(bomb.waveLeft[2*i], bomb.waveLeft[2*i+1], \
        32, 32).collidelistall(tempBlockRects):
        break
    for i in range(bomb.powerLvl):
        bomb.waveRight += [bombsPlaced[0].x+(32*(i+1)), \
            bombsPlaced[0].y]
        if pygame.Rect(bomb.waveRight[2*i], bomb.waveRight[2*i+1], 32, \
            32).collidelistall(tempBlockRects):
            break
    for i in range(bomb.powerLvl):
        bomb.waveUp += [bombsPlaced[0].x, \
            bombsPlaced[0].y-(32*(i+1))]
        if pygame.Rect(bomb.waveUp[2*i], \
            bomb.waveUp[2*i+1], \
            32, 32).collidelistall(tempBlockRects):
            break
    for i in range(bomb.powerLvl):
        bomb.waveDown += [bombsPlaced[0].x, \
            bombsPlaced[0].y+(32*(i+1))]
        if pygame.Rect(bomb.waveDown[2*i], bomb.waveDown[2*i+1], \
            32, 32).collidelistall(tempBlockRects):
            break

    for j in range( int(len(bomb.waveLeft)/2) ):
        bomb.waveDestruction(bomb.waveLeft[j*2], \
            bomb.waveLeft[j*2+1])
    for j in range( int(len(bomb.waveRight)/2) ):
        bomb.waveDestruction(bomb.waveRight[j*2], \
            bomb.waveRight[j*2+1])
    for j in range( int(len(bomb.waveUp)/2) ):
        bomb.waveDestruction(bomb.waveUp[j*2], \ bomb.waveUp[j*2+1])
    for j in range( int(len(bomb.waveDown)/2) ):
        bomb.waveDestruction(bomb.waveDown[j*2], \
            bomb.waveDown[j*2+1])
    bomb.waveItself = [bombsPlaced[0].x, bombsPlaced[0].y]
    bomb.waveDestruction(bomb.waveItself[0], bomb.waveItself[1])
    bomb.waveExists = True
    bombsPlaced.pop(0)

def waveDestruction(X, Y):
    tempBlockRects = []
    tempEnemyRects = []

    waveRect = pygame.Rect(X, Y, 32, 32)
    for j in range(len(blocksList)):
        tempBlockRects += [pygame.Rect(blocksList[j].hitbox)]
    for j in range(len(enemiesList)):
        tempEnemyRects += [pygame.Rect(enemiesList[j].hitbox)]

    if waveRect.collidelistall(tempBlockRects):
        if blocksList[ waveRect.collidelistall\
            (tempBlockRects)[0] ].entityType in ['2', '3']:
            blocksList.pop( waveRect.collidelistall\
                (tempBlockRects)[0] )
    if waveRect.collidelistall(tempEnemyRects):
        pygame.mixer.Sound.play(enemyDeadSounds[lvlNum])
        enemiesList.pop( .collidelistall\ (tempEnemyRects)[0] )
    if waveRect.colliderect(pygame.Rect(bomberMan.hitbox)):
        bomberMan.HIT = True

waveTimer = 0
def bombWaveDraw(win):
    if bomb.waveTimer +1 >= 9:
        bomb.waveTimer = 0
        bomb.waveExists = False
    else:
        bomb.waveTimer += 1

```



```

win.blit(bombItselfBoom[0], (bomb.waveItself[0],\
    bomb.waveItself[1]))
for i in range( int(len(bomb.waveLeft)/2) ):
    win.blit(pygame.transform.rotate\
        (bombWaves[bomb.waveTimer], 90),\
        (bomb.waveLeft[2*i], bomb.waveLeft[2*i+1]))
for i in range( int(len(bomb.waveRight)/2) ):
    win.blit(pygame.transform.rotate\
        (bombWaves[bomb.waveTimer], 270),\
        (bomb.waveRight[2*i], bomb.waveRight[2*i+1]))
for i in range( int(len(bomb.waveUp)/2) ):
    win.blit(bombWaves[bomb.waveTimer],\
        (bomb.waveUp[2*i], bomb.waveUp[2*i+1]))
for i in range( int(len(bomb.waveDown)/2) ):
    win.blit(pygame.transform.rotate\
        (bombWaves[bomb.waveTimer], 180), (bomb.waveDown[2*i],\
        bomb.waveDown[2*i+1]))

class player(object):
    remaininglives = 3
    def __init__(self, x, y, width, height):
        self.HIT = False
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.vel = 5
        self.left = False
        self.right = False
        self.up = False
        self.down = False
        self.walkCount = 0
        self.standing = True
        self.hitbox = (self.x+8, self.y+8, self.width-16,self.height -16)
        self.boxL = (self.x, self.y, 1, 31)
        self.boxR = (self.x+31, self.y, 1, 31)
        self.boxU = (self.x, self.y, 31, 1)
        self.boxD = (self.x, self.y+31, 31, 1)

    def draw(self, win, walkLeft, walkRight, walkUp, walkDown):
        if self.walkCount + 1 >= 27:
            self.walkCount = 0

        if not(self.standing):
            if self.left:
                win.blit(walkLeft[self.walkCount//3], (self.x, self.y))
                self.walkCount += 1
            elif self.right:
                win.blit(walkRight[self.walkCount//3], (self.x, self.y))
                self.walkCount += 1
            elif self.up:
                win.blit(walkUp[self.walkCount//3], (self.x, self.y))
                self.walkCount += 1
            elif self.down:
                win.blit(walkDown[self.walkCount//3], (self.x, self.y))
                self.walkCount += 1
        else:
            if self.left:
                win.blit(walkLeft[0], (self.x, self.y))
            elif self.right:
                win.blit(walkRight[0], (self.x, self.y))
            elif self.up:
                win.blit(walkUp[0], (self.x, self.y))
            else:
                win.blit(walkDown[0], (self.x, self.y))
        #we move => hitbox moves
        self.hitbox = (self.x+8, self.y+8, self.width-16,self.height -16)

```

```

self.boxL = (self.x+8, self.y+8, 1, self.height - 16)
self.boxR = (self.x+self.width-8, self.y+8, 1, self.height-16)
self.boxU = (self.x+8, self.y+8, self.width-16, 1)
self.boxD = (self.x+8, self.y+self.height-8, self.width-16, 1)
self.collision()

def collision(self):
    tempEnemyRects = []
    for j in range(len(enemiesList)):
        tempEnemyRects += [pygame.Rect(enemiesList[j].hitbox)]
    for ENEMY in tempEnemyRects:
        if pygame.Rect(bomberMan.hitbox).colliderect(ENEMY):
            self.HIT = True

class enemy(object):
    def __init__(self, x, y, width, height, velX, velBonus, enemyType):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.walkCount = 0
        self.velX = velX
        self.velY = 0
        self.velDefault = 5
        self.velBonus = velBonus
        self.hitbox = (self.x+10, self.y+10, self.x + self.width -20,\
            self.y + self.height - 20)
        self.boxL = (self.x, self.y, 1, 31)
        self.boxR = (self.x+31, self.y, 1, 31)
        self.boxU = (self.x, self.y, 31, 1)
        self.boxD = (self.x, self.y+31, 31, 1)
        self.eventCounter = 0
        self.eventAction = 0
        self.enemyType = enemyType
        self.walkLeft = []
        self.walkUp = []
        self.walkDown = []

    def draw(self, win):
        self.getEnemySprites() #different for every enemyType
        self.moveLogic() #each time we draw we also move ur enemy
        if self.walkCount + 1 >= 27:
            self.walkCount = 0
        if self.velX < 0: #on x - moving left
            win.blit(self.walkLeft[self.walkCount//3], (self.x, self.y))
            self.walkCount += 1
        elif self.velX > 0: #on x - moving right
            win.blit(pygame.transform.flip\
                (self.walkLeft[self.walkCount//3], 1, 0), (self.x, self.y))
            self.walkCount += 1
        elif self.velY < 0: #on y - moving up
            win.blit(self.walkUp[self.walkCount//3], (self.x, self.y))
            self.walkCount += 1
        elif self.velY > 0: #on y - moving down
            win.blit(self.walkDown[self.walkCount//3], (self.x, self.y))
            self.walkCount += 1
        else:
            win.blit(self.walkDown[0], (self.x, self.y))
            self.walkCount = 0

        self.boxL = (self.x+5, self.y+16, 1, 2)
        self.boxR = (self.x+27, self.y+16, 1, 2)
        self.boxU = (self.x+16, self.y+5, 2, 1)
        self.boxD = (self.x+16, self.y+27, 2, 1)
        self.hitbox = (self.x, self.y, self.width, self.height)

```

```

def moveLogic(self):
    if self.enemyType == '7':
        self.velDefault = 5
    elif self.enemyType == '9':
        self.velDefault = 1
    elif self.enemyType == '8':
        self.velDefault = 6

    tempBlockRects = []
    for j in range(len(blocksList)):
        tempBlockRects += [pygame.Rect(blocksList[j].hitbox)]
    collisionBoxL = \
    (pygame.Rect(self.boxL).collidelistall(tempBlockRects))
    collisionBoxR = \
    (pygame.Rect(self.boxR).collidelistall(tempBlockRects))
    collisionBoxU = \
    (pygame.Rect(self.boxU).collidelistall(tempBlockRects))
    collisionBoxD = \
    (pygame.Rect(self.boxD).collidelistall(tempBlockRects))
#we're moving RIGHT
    if self.velX > 0 and not collisionBoxR:
        self.x += self.velX
    #we're moving LEFT
    elif self.velX < 0 and not collisionBoxL:
        self.x += self.velX
    #we're moving DOWN
    elif self.velY > 0 and not collisionBoxD:
        self.y += self.velY
    #we're moving UP
    elif self.velY < 0 and not collisionBoxU:
        self.y += self.velY

    self.eventCounter += 1
    if self.eventCounter > 54:
        self.eventAction = random.randint(-1,1)
        self.eventCounter = 0
    if collisionBoxR:
        self.x -= 4
        self.velY = self.velX*(-1)**random.randint(0,1)
        self.velX = 0
    if collisionBoxL:
        self.x += 4
        self.velY = self.velX*(-1)**random.randint(0,1)
    self.velX = 0
    if collisionBoxD:
        self.y -= 4
    self.velX = self.velY*(-1)**random.randint(0,1)
    self.velY = 0
    elif collisionBoxU:
        self.y += 4
        self.velX = self.velY*(-1)**random.randint(0,1)
        self.velY = 0
    if self.eventAction == -2:
        if self.velX == 0 and self.velY == 0:
            self.velX = self.velDefault*(-1)**random.randint(0,1)
    elif self.eventAction == -1:
        self.velY = -self.velBonus*(-1)**random.randint(0,1)-1
    else: #self.eventAction == 0:
        if self.velX == 0 and self.velY == 0:
            self.velY = self.velDefault *(-1)**random.randint(0,1)

def getEnemySprites(self):
    if self.enemyType == '7':
        self.walkLeft, self.walkUp, self.walkDown = wolfWalkLeft,\
        wolfWalkUp, wolfWalkDown
    elif self.enemyType == '9':
        self.walkLeft, self.walkUp, self.walkDown = slimeWalkLeft,\
        slimeWalkUp, slimeWalkDown
    elif self.enemyType == '8':

```

```

        self.walkLeft, self.walkUp, self.walkDown = batWalkLeft,\
        batWalkUp, batWalkDown

def redrawGameWindow():
    win.blit((levelMap[lvlNum]), (0,-64))
    bomberMan.draw(win, playerWalkLeft, playerWalkRight, playerWalkUp,\
        playerWalkDown)
    for BOMB in bombsPlaced:
        BOMB.draw(win)
    for ENEMY in enemiesList:
        ENEMY.draw(win)
    for ENTITY in blocksList:
        ENTITY.draw(win)
    if bomb.waveExists:
        bomb.bombWaveDraw(win)
    pygame.display.update()

def MAPLOAD():
    global blocksList, enemiesList
    for i in range(len(entityTypes)):
        for j in range(len(entityTypes)):
            randomValue = random.randint(0,1)
            if entityTypes[i][j] in ['1','3']:#blocks
                blocksList +=[entity(32*j,32*i,32,32,entityTypes[i][j] )]
            elif entityTypes[i][j] == '2' and randomValue == 0:
                blocksList +=[entity(32*j,32*i,32,32,entityTypes[i][j] )]
            elif entityTypes[i][j] == '7':
                enemiesList+=[enemy(32*j,32*i,32,32,5,10,\
                    entityTypes[i][j])]
            elif entityTypes[i][j] == '9':
                enemiesList+=[enemy(32*j,32*i,32,32,2,0,\
                    entityTypes[i][j])]
            elif entityTypes[i][j] == '8':
                enemiesList+=[enemy(32*j,32*i,32,32,6,6,\
                    entityTypes[i][j])]

def NEWMAP(currentLvl):
    global lvlNum
    lvlNum = currentLvl
    entityTypes[:] = []
    blocksList[:] = []

def calcPlayerSuperPos(XorY):
    if XorY == 'X':
        bombSuperPosX = 0
        if (bomberMan.x % 32) < 16:
            bombSuperPosX = bomberMan.x - (bomberMan.x % 32)
        else:
            bombSuperPosX = bomberMan.x + (32 - (bomberMan.x % 32))
        return bombSuperPosX
    elif XorY == 'Y':
        bombSuperPosY = 0
        if (bomberMan.y % 32) < 16:
            bombSuperPosY = bomberMan.y - (bomberMan.y % 32)
        else:
            bombSuperPosY = bomberMan.y + (32 - (bomberMan.y % 32))
        return bombSuperPosY

pygame.mixer.pre_init(44100, -16, 2, 512)
pygame.mixer.init()
pygame.init()

screenWidth, screenHeight = 1000, 700

```

```

win = pygame.display.set_mode( (screenWidth, screenHeight) )
pygame.display.set_caption("BomberMan TIM")

entityTypes = []
blocksList = []
enemiesList = []
bombsPlaced = []
bomberMan = None
lvlNum = None

playerWalkLeft = [pygame.image.load\
('data\sprites\player\playerL'+str(i)+'.png') for i in range(1,10)]
playerWalkRight = [pygame.image.load\
('data\sprites\player\playerR'+str(i)+'.png') for i in range(1,10)]
playerWalkUp = [pygame.image.load\
('data\sprites\player\playerU'+str(i)+'.png') for i in range(1,10)]
playerWalkDown = [pygame.image.load\
('data\sprites\player\playerD'+str(i)+'.png') for i in range(1,10)]
wolfWalkLeft = [pygame.image.load\
('data\sprites\wolf\wolfL'+str(i)+'.png')\
for i in range(1,10)]
wolfWalkUp = [pygame.image.load('data\sprites\wolf\wolfU'+str(i)+'.png')\
for i in range(1,10)]
wolfWalkDown = [pygame.image.load\
('data\sprites\wolf\wolfD'+str(i)+'.png')\
for i in range(1,10)]
slimeWalkLeft = [pygame.image.load\
('data\sprites\slime\slimeL'+str(i)+'.png') for i in range(1,10)]
slimeWalkUp = [pygame.image.load\
('data\sprites\slime\slimeU'+str(i)+'.png')\
for i in range(1,10)]
slimeWalkDown = [pygame.image.load\
('data\sprites\slime\slimeD'+str(i)+'.png') for i in range(1,10)]
batWalkLeft = [pygame.image.load('data\sprites/bat/batL'+str(i)+'.png')\
for i in range(1,10)]
batWalkUp = [pygame.image.load('data\sprites/bat/batU'+str(i)+'.png')\
for i in range(1,10)]
batWalkDown = [pygame.image.load('data\sprites/bat/batD'+str(i)+'.png')\
for i in range(1,10)]
bombPlaced = [pygame.image.load('data\items/bomb'+str(i)+'.png')\
for i in range(1,10)]
bombItselfBoom = [pygame.image.load('data\items/bombBoom.png')]
bombWaves = [pygame.image.load('data\items/wave'+str(i)+'.png')\
for i in range(1,10)]
levelMap = [pygame.image.load('data\levels\lvl'+str(i)+'.png')\
for i in range(0,3)]
l0BBlock = pygame.image.load('data\items\BBlock.png')
l0UBlock = pygame.image.load('data\items\UNBBlock.png')

pygame.mixer.music.load("data/backGroundMusic.mp3")
pygame.mixer.music.set_volume(0.3)
bombPlaceSound = pygame.mixer.Sound('data/bombPlaced.ogg')
bombPlaceSound.set_volume(0.3)
bombBoomSound = pygame.mixer.Sound("data/bombExplosion.ogg")
bombBoomSound.set_volume(0.8)
enemyDeadSounds = [pygame.mixer.Sound\
("data/enemyDeathSound"+str(i)+".ogg") for i in range(0,3)]

```