Министерство образования Республики Беларусь

Белорусский Национальный Технический Университет

Факультет информационных технологий и робототехники

Кафедра «программное обеспечение вычислительной техники и автоматизированных систем»

**Курсовой проект**

**по дисциплине «Объектно-ориентированные технологии программирования и СП»**

Тема: « Игровое 2D-приложение ”Mario” »

Исполнитель: Студент группы 1070115

Куропатин Павел Владимирович

Руководитель: Иванченко Виктор Викторович

Минск 2016

# Содержание

[Содержание 4](#_Toc468700838)

[Введение 5](#_Toc468700839)

[1 Техническое задание на предмет разработки 7](#_Toc468700840)

[2 Обоснование используемых языков и технологий 9](#_Toc468700841)

[3 Моделирование и проектирование программного обеспечения 12](#_Toc468700842)

[4 Реализация программного обеспечения 21](#_Toc468700843)

[5 Развертывание и тестирование программного обеспечения 23](#_Toc468700844)

[Заключение 25](#_Toc468700845)

[Список использованных источников 26](#_Toc468700846)

[Приложение 27](#_Toc468700847)

[**Приложение А** 27](#_Toc468700848)

[**Приложение Б** 28](#_Toc468700849)

[**Приложение В** 30](#_Toc468700850)

[**Приложение Г** 32](#_Toc468700851)

[**Приложение Д** 34](#_Toc468700852)

[**Приложение Е** 35](#_Toc468700853)

[**Приложение Ж** 36](#_Toc468700854)

# Введение

В современном мире разработка компьютерных игр является одно из популярных направлений среди разработки программного обеспечения. Существует множество языков программирования, способных для разработки игр.

Игры помогают нам смоделировать различные жизненные ситуации, проблемы и выдают некоторые возможные пути их решения. Игра содержит в себе все необходимые предпосылки для естественного развития личности и культуры общества.

Вместе с появлением компьютеров появились компьютерные игры, которые сразу же нашли массу поклонников. . С совершенствованием компьютеров совершенствовались и игры, привлекая все больше и больше людей. На сегодняшний день компьютерная техника достигла такого уровня развития, что предоставляет возможность программистам разрабатывать очень реалистичные игры с хорошим графическим и звуковым оформлением.[1]

В данной пояснительной записке содержатся следующие главы:

1. Техническое задание на предмет разработки, содержащее описание весь функционал, который необходимо разработать, а также перечень используемых технологий и языков программирования
2. Обоснование используемых языков и технологий, дающее понять почему были использованы данные языки и технологии, их достоинства, которые были использованы в данном проекте
3. Моделирование и проектирование программного обеспечения, содержащее в себе структуру разрабатываемого проекта, объекты используемые в данном проекте, а также UML-диаграммы отношений между объектами
4. Реализация программного обеспечения, описывающая процесс разработки и демонстрирующая плюсы используемых технологий на примере проекта.
5. Развертывание и тестирование. Содержит в себе методику и план тестирования основного функционала и всё окружение необходимое для разворачивания приложения
6. Заключения – краткий итог проделанной вышеописанной работы

Цель курсового проекта является закрепление основ объектно-ориентированного программирование, получение навыка в разработки простых 2D-приложений, а также создание приложения благодаря, которому можно реализовать полезное времяпровождение.

# 1 Техническое задание на предмет разработки

Разработка данного приложения «Марио» включает в себя разработку следующее окружение:

1. Язык программирования Python
2. Framework Pygame
3. Модуль Piganim

Также разработка данного приложения подразумевает разработку следующего набора функций. В этот набор входит:

1. Создание окна игры
2. Разработка пользовательского интерфейса в виде меню, с определенными возможностями
   1. Для меню при запуске приложения соответствует меню со следующими пунктами: выбор количество игроков, настройка конфигурации управления, выход из игры
   2. Для меню во время прохождения игры соответствует меню со следующими пунктами: настройка конфигурации управления, завершение текущей игры, возврат в активную игру, выход из приложения
3. Передвижения героя
4. Передвижение монстров
5. Смена изображений монстров и героя игры.
6. Реакция героя на бонусы.
7. Организация генератора уровня
8. Разрушаемость определенных блоков.
9. Переход на следующий уровень.
10. Отображение состояние уровня, включающее в себя номер уровня, отображение жизней героя, время действия бонусов.
11. Отображение объектов во время игры.
12. Передвижение окружающих героя объектов при движении героя.
13. Организация генератора уровня
14. Разработка действий, выполняемых при смерти героя или монстра.

Среди персонажей требуется разработать:

1. Герой (Hero) – обладает способностью движения, 3 жизни. Изображение меняется в зависимости от направления движения. При подбирании бонус «цветок» увеличивается сила прыжка на время действия бонуса, при истечении действия бонус прыжок устанавливается стандартный. При подбирании бонуса «гриб» приобретает способность стрелять во врагов, по истечении времени действия бонуса способность отменяется
2. Простой монстр (Slub) – обладает простейшим движением, 1 жизнь. Убивается выстрелом или прыжком героя сверху
3. Сложный монстр (Bowser) – обладает простым движением, 5 жизней. Убивается только выстрел в лоб.
4. Монстр-цветок (BlueFlower) – движение вверх-вниз, 1 жизнь, появляется из трубы. Убивается выстрелом с любой стороны.

Среди персонажей требуется разработать:

1. Платформа – платформа, опора, по которым передвигаются персонажи, не разрушаемая.
2. Простой блок – вид платформы, разрушается от 2х ударов снижу персонажем
3. Бонус-блок – платформа, при первом ударе героем снизу появляется бонус, после платформа становится простой, не разрушаемая.
4. Труба – платформа, из которой появляется Монстр-цветок, не разрушаемая.

# 2 Обоснование используемых языков и технологий

Игра «Марио» сама по себе является платформером. Платформер - жанр компьютерных игр и видеоигр, где основной чертой игрового процесса является прыгание по платформам, лазанье по лестницам, собирание предметов, обычно необходимых для завершения уровня. Некоторые предметы, называемые пауэр-апами (англ. power-up), наделяют управляемого игроком персонажа особой силой, которая обычно иссякает со временем (к примеру: силовое поле, ускорение, увеличение высоты прыжков). Противники (называемые «монстрами» независимо от внешнего вида), всегда многочисленные и разнородные, обладают примитивным искусственным интеллектом, стремясь максимально приблизиться к игроку, либо не обладают им вовсе, перемещаясь по круговой дистанции или совершая повторяющиеся действия. Соприкосновение с противником обычно отнимает жизненные силы у героя или вовсе убивает его. Иногда противник может быть нейтрализован либо прыжком ему на голову, либо из оружия, если им обладает герой. Смерть живых существ обычно изображается упрощённо или символически (существо исчезает или проваливается вниз экрана).[2]

Исходя из вышеописанного жанра разрабатываемой игры, можно прийти к выводу, что наиболее подходящими технологиями являются следующие:

1. Язык программирования Python
2. Framework Pygame
3. Модуль Piganim
4. Потоки.

Язык программирования Python

Python – это один из наиболее популярных современных языков программирования. Он пригоден для решения разнообразных задач и предлагает те же возможности, что и другие языки программирования: динамичность, поддержку ООП и кросс-платформенность.

Если программист только начинает свой путь в области разработки ПО, то Python станет идеальным «вводным» языком программирования. Благодаря своей лаконичности он позволит быстрее овладеть синтаксисом языка, а отсутствие «наследства» в виде формировавшихся на протяжении многих лет аксиом и принципов разработок ПО поможет быстро освоить ООП. В силу этих факторов «кривая обучения» Python может быть довольно

короткой, и программист сможет перейти от учебных примеров к коммерческим проектам.

Так как язык программирования Python является скриптовым, то он является больше производительным, что является одним из основных причин и плюсов его использования в данном проекте.

Использование потоков

Существуют две самые распространенные причины использовать потоки: во-первых, для увеличения эффективности использования многоядерной архитектуры современных процессоров, а значит, и производительности программы; во-вторых, если нам нужно разделить логику работы программы на параллельные полностью или частично асинхронные секции.

В первом случае мы сталкиваемся с таким ограничением, как Global Interpreter Lock. Концепция GIL заключается в том, что в каждый момент времени только один поток может исполняться процессором. Это сделано для того, чтобы между потоками не было борьбы за отдельные переменные. Исполняемый поток получает доступ по всему окружению. Такая особенность реализации потоков в Python значительно упрощает работу с потоками и дает определенную потокобезопасность.

Второй случай имеет место в нашем проекте, поскольку потоком будет запускаться «искусственный интеллект» монстров.

Framework Pygame и модуль Pyganim

Pygame, естественно, не рассчитан для сложных приложений с 2D графикой и очень тяжёлым геймплеем. Хоть т игры на движке pygame сейчас почти невостребованные (по перечисленным выше причинам), но будут хорошей практикой для начинающего игрового программиста. Делать игры на Python — это разминка перед серьёзной разработкой игр и приложений.

Pygame — отличный набор библиотек, имея на руках которые, можно написать несложную игру. Но его возможности предельно ограничены и Pygame пригоден исключительно для тренировки и интерактивного изучения языка Python.

Pyganim представляет собой модульный основанный на фреймфорке Pygame. С помощью Pyganim можно качественно анимировать игровые модели и спрайты, таким образом сделав игру красивее и соответственно, более востребованной в современной разработке игр.

# 3 Моделирование и проектирование программного обеспечения

Структура проекта, в соответствии с применяемым паттерном MVC, состоит из трёх компонент: Model, View, Controller и пользовательский пакет Utility. В ходе разработки приложения возникла необходимость более структурированного проекта для его поддержания. Для достижения этой цели пакет Entity был «разбит» на две части: «статическую» и «динамическую».

Статическая часть – пакет, в котором содержатся классы объектов, которые не двигаются, т.е. статичны. Динамическая часть соответственное содержит такие классы объектов, как Герои, Монстры и т.п., которые способны передвигаться. Для выше описанной необходимости более структурированного проекта динамическая часть была разделена на Героев и Монстров.

В данном проекте вышеуказанные части имеют название:

1. Статическая часть – dead
2. Динамическая часть – alive
   1. Герои – heroes
   2. Монстры – enemies

Конечная структура приложения приведена на рисунке 3.1 (ниже) без самих классов.

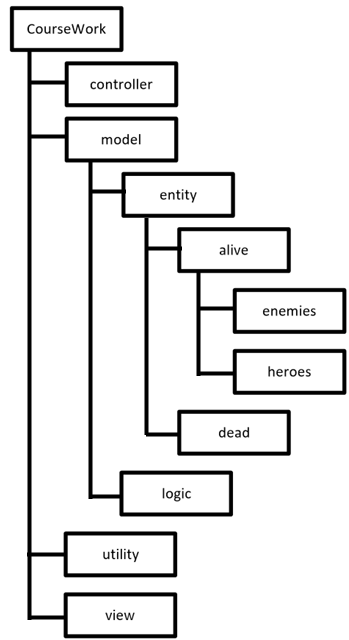


Рисунок 3.1 – Структура проекта

Для сопоставления вышеописанной структуры к классам приложения были реализованные UML-даиграммы. Среди раззработанных диаграмм присутствуют:

1. Диаграмма иерархии «статической части» (см. рисунок 4.2)
2. Диаграмма иерархии «динамической части» (см. рисунок 4.3)
3. Диаграмма утилитного класса Camera (см. рисунок 4.4)
4. Диаграмма класса отображения ViewBackground (см. рисунок 4.5)
5. Диаграмма класса генератора уровня LevelGenerator (см. рисунок 4.6)
6. Диаграммы классов-логик игры (см. рисунок 4.7-4.10)



Рисунок 4.2 – UML-диаграмма иерархии «статических» объектов



Рисунок 4.3 – UML-диаграмма иерархии «динамических» объектов



Рисунок 4.4 – UML-диаграмма класса Camera



Рисунок 4.5 – UML-диаграмма класса ViewBackground



Рисунок 4.6– UML-диаграмма класса-генератора LevelGenerator



Рисунок 4.7 – UML-диаграмма класса-логики LogicMonsterr



Рисунок 4.8 – UML-диаграмма класса-логики LogicGame



Рисунок 4.9 – UML-диаграмма класса-логики LogicHero



Рисунок 4.10 – UML-диаграмма класса-логики LogicMenu

Коды классов логики и класса ViewBackground приведены в приложении А-Д.

Сценарий данной игры-платформера подразумевают следующие действия:

1. Меню игры, содержащее в себе выбор количества игроков, настройки управления и выход из приложения. Меню также может быть вызвано во время игры.
2. При запуске самой игры персонаж появляется в начальной точке.
3. Уровень считается пройденным, когда персонаж дойдет до точки перехода на другой уровень. При прохождении последнего уровня появится сообщения о удачном завершении игры.
4. Во время прохождения на пути персонажа появляются монстры, обладающие определенными способностями, также существуют бонусы, дающие герою определённые возможности

Примерное оформление игры изображено на рисунке 1.1, приведенном ниже.

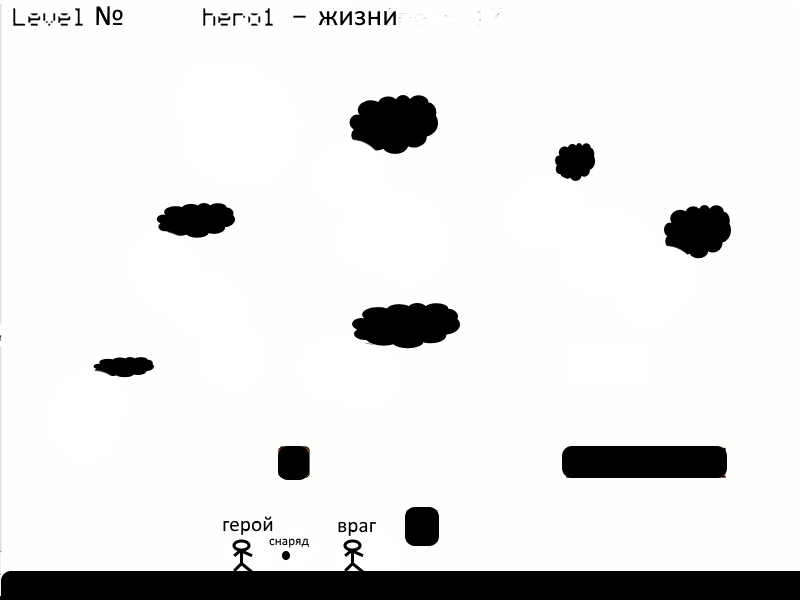


Рисунок 2.1

# 4 Реализация программного обеспечения

На этапе основного производства выполняется огромный объём работ. Был разработан исходный код, создана графика (спрайты). Также созданы были звуковые эффекты, отредактированы уровни. Всё это время также велось дополнение и изменение игрового дизайна, чтобы отразить текущее видение игры.

Весь графический исользуемый материал отображен в Приложении (см. Приложение Е).

В начале разработки игры требовалось разработать функции отображения «статических» объектов. Для этого во фрейворке Pygame существует класс Surface, принимающий в параметре размер объекта и являющийся собой фоном для изображения, класс Rect, среди аргументов, которого имеются координаты объекта относительно уровня, функция Pygame-а blit, прорисовывающая изображение объект по его координатам относительно уровня и естественно функция «обновления» экрана.

Как и в любой игре, необходим обработчик нажатия клавиш. Фреймворк Pygame предоставляет данную возможность. Она реализуется с помощью метода event.get (). Этот метод позволяет определить тип действия томительно клавиатуры и компьютерной мыши, например, нажатие клавиши, отпускание нажатой клавиши, левый и правый щелчок мыши. Клавиши в Pygame представлены в виде «K\_(название\_клавиши)». Pygame возвращает Ascii-коды этих клавиш, что значительно упрощает работу с клавиатурой.

Для смены изображения был использован ранее описанный модуль Pyganim, основанный на фреймфорке Pygame. Этот модуль обеспечивает быструю смену изображений героя в соответствии с нажатой клавишей.

Для обработки взаимодействия объектов были использованы два потока: один – для обработки действий героя, второй – для обработки действий персонажа. Как и говорилось ранее, поток используется для параллельных процесса. Это можно объяснить следующим, игра представляет собой бесконечный цикл, если использовать обычные функции, то игра будет зависать на 1-2 секунды пока обработчик не пройдется по всем объектам. Поскольку потоки выполняют функции с циклом, то можно быть уверенным, что поток не будет запущен или действовать автономно.

Для создания меню были применены вышеописанный обработчик нажатия клавиш, метод Pygame-a для отрисовки текстовой информации render, метод Pygame-a Font, определяющий стиль и шрифт текста отрисовки.

Для генератора уровней используются текстовые файлы уровней, в которых содержатся схема уровня в виде символов, которым соответствуют определенный тип объекта в игре.

Конечное изображение уровня отображено в Приложении (см. Приложение Ж)

# 5 Развертывание и тестирование программного обеспечения

Для развертывания программного обеспечения использовалось следующее окружение. Во-первых, естественно сам Python. Его можно легко установить. Во-вторых, необходимые пакеты pygame и py2exe. Их можно установить через командную строку командой pip:

…> pip install <имя пакета>

Пакет py2exe используется для преобразования исходных python-файлов проекта в единый exe-файл. Отдельная от проекта существует файл setup.py, который является в определенном смысле установщиком игры. Для установки игры в командной строке прописывается следующая команда:

…> python setup.py py2exe

После выполнения команды в каталоге с проектом содержится папка dist, содержащая в себе exe-файл приложения. При этом вышеописанная папка dist является самим приложением, приспособленным к распространению.

Тестирование приложения. План тестирования:

1. Объект тестирования – приложение «Mario»
2. Функции тестирования - основной функционал игры, включающий в себя передвижения героя, контакт с объектами и бонусами, переход на следующий уровень
3. Вид тестирования – функциональный

Для данного приложения выбрана методика под названием «сверху-вниз». Этот метод тестирования является дополнительным этапом процесса проектирования сверху вниз, сквозного контроля и кодирования сверху вниз. При таком методе разработки программного обеспечения вначале пишется основная программа. По мере того как "скелет" программы "обрастает" новыми модулями, должны добавляться и новые тестовые данные, объем которых увеличивается постепенно, одновременно с разрастанием программы. В результате появляется возможность накапливать тестовые данные вместо раздельного их формирования для каждого модуля. Еще одним плюсом тестирования по методу сверху вниз является то, что стержневая логика программы тестируется на раннем этапе, и эта проверка повторяется многократно с добавлением новых модулей. [3]

В ходе тестирования всего функционала во все случаях были достигнуты следующие результаты:

* Точное передвижение персонажа во всех направлениях
* Переход на следующий уровень
* Завершение игры(успешное и нет)
* Разрушаемость блоков
* Отображение всех блоков
* Передвижение монстров
* Стрельба
* Отображение меню во время игры и стартовое меню, а также все пункты, перечисленные в каждом меню
* Изменение конфигурации управления
* Воспроизведение звука

В ходе тестирования всего функционала были обнаружены следующие ошибки (баги):

* Исчезновение монстров в связи с их смертью (в 2 тестах 20)
* Неполноценная прорисовка бонусов (в 3 тестах 20)

# Заключение

Программа была разработана для интересного и полезного времяпровождения. Целью проекта была разaработка 2D- приложения «Марио» , который мы добились. В процессе тестирования были выявлены ошибки, большая часть которых были исправлены.

В процессе разработки приложения были закреплены основы объектно-ориентированного программирования в Python, а также самостоятельно изучены:

1. Фреймворк Pygame
2. Модуль Pyganim
3. Графический редактор Paint.net

Благодаря используемым технология приложения способно масштабироваться, расширяться, а также дополняться новым функционалом, например более сложные движения героев, более усложнённый сюжет и т.п.. Также планируется вместо текстовых файлов со схематичным отображением уровня применять Tiled Map Editor, представляющий собой редактор уровней платформенных игр и способный создавать «слоёные» уровни, что обеспечит более упрощенную генерацию уровня.[4]

# Список использованных источников

1. Интернет-ресурс: <https://books.google.by/books?id=84TVBQAAQBAJ&printsec=frontcover&hl=ru> Дата доступа: 29.11.2016
2. Платформер. Интернет-ресурс: <http://make-games.ru/sozdat_igru>. Дата доступа :29.11.2016.
3. Методика тестирования. Интернет-ресурс: <http://pazl.biz/razrabotka-i-otladka-programm/59-metodi-testirovaniya.html> Дата доступа: 14.11.2016.
4. Tilde Map Editor. Интернет –ресурс: <http://www.mapeditor.org/>. Дата доступа: 27.11.2016

# Приложение

**Приложение А**

**Код класса Camera**

from pygame import Rect

class Camera:

def \_\_init\_\_(self, width=0, height=0, window\_width=0, window\_height=0):

self.\_\_window\_width = window\_width

self.\_\_window\_height = window\_height

self.\_\_window = Rect(0, 0, width, height)

@property

def window(self):

return self.\_\_window

@window.setter

def window(self,value\_rect):

self.\_\_window = value\_rect

@property

def window\_width(self):

return self.\_\_window\_width

@window\_width.setter

def window\_width(self, value):

self.\_\_window\_width = value

@property

def window\_height(self):

return self.\_\_window\_height

@window\_height.setter

def window\_height(self, value):

self.\_\_window\_height = value

def shift\_object(self, object\_rect):

return object\_rect.move(self.window.topleft)

def in\_center(self, object\_rect):

left, top, \_, \_ = object\_rect

\_, \_, width\_rect, height\_rect = self.window

left, top = -left + self.window\_width // 2, -top + self.window\_height // 2

left = min(0, left) - 32

left = max(-(self.window.width - self.window\_width), left)

top = max(-(self.window.height - self.window\_height), top)

top = min(0, top)

self.window = Rect(left, top, width\_rect, height\_rect)

**Приложение Б**

**Код класса LogicGame**

import pygame

from pygame import \*

import threading

import sys, os

from model.entity.Camera import Camera

from model.entity.LevelGenerator import LevelGenerator

from model.entity.alive.heroes.Hero import Hero

from model.logic.LogicHero import LogicHero

from model.logic.LogicMenu import LogicMenu

from model.logic.LogicMonster import LogicMonster

from view.ViewBackgound import ViewBackground

from model.entity.alive.heroes.Heroes import Heroes

from model.entity.alive.enemies.BlueFlower import BlueFlower

import random

class LogicGame:

@staticmethod

def start\_game\_window(background, background\_color, window, timer, during\_game=False):

settings = [ord('w'), ord('a'), ord('d'), ord('f'), ord('i'), ord('j'), ord('l'), ord(';')]

LogicMenu.font = "data/fonts/coders\_crux.ttf"

LogicMenu.height\_menu = 60

LogicMenu.height\_game = 40

LogicMenu.background\_color = background\_color

LogicMenu.color\_active\_text = (250, 30, 250)

LogicMenu.color\_simple\_text = (250, 250, 30)

while True:

choice = LogicMenu.menu(background, window, during\_game=False)

if choice == 1 or choice == 2:

''' Creating Heroes'''

heroes = Heroes()

hero1 = Hero(x=170, y=500, width=22, height=31, keys=settings[:4], type="hero1", animation\_koef=3,

animation\_right=['data/mario/r1.png', 'data/mario/r2.png', 'data/mario/r3.png',

'data/mario/r4.png', 'data/mario/r5.png'],

animation\_left=['data/mario/l1.png', 'data/mario/l2.png', 'data/mario/l3.png',

'data/mario/l4.png', 'data/mario/l5.png'],

animation\_jump\_left='data/mario/jl.png', animation\_jump\_right='data/mario/jr.png',

animation\_jump='data/mario/j.png', side=False, power=1, lifes=3,

time\_flower\_activity=0, time\_mushroom\_activity=0, left=False, right=False, up=False,

on\_ground=False, xvel=0, yvel=0, move\_speed=4, jump\_power=16, gravity=1,

fire\_ability=False, flower\_ability=False)

heroes.append(hero1)

if choice == 2:

hero2 = Hero(x=270, y=170, width=22, height=31, keys=settings[4:], type="hero2", animation\_koef=3,

animation\_right=['data/mario/r1.png', 'data/mario/r2.png', 'data/mario/r3.png',

'data/mario/r4.png', 'data/mario/r5.png'],

animation\_left=['data/mario/l1.png', 'data/mario/l2.png', 'data/mario/l3.png',

'data/mario/l4.png', 'data/mario/l5.png'],

animation\_jump\_left='data/mario/jl.png', animation\_jump\_right='data/mario/jr.png',

animation\_jump='data/mario/j.png', side=False, power=1, lifes=3,

time\_flower\_activity=0, time\_mushroom\_activity=0, left=False, right=False, up=False,

on\_ground=False, xvel=0, yvel=0, move\_speed=4, jump\_power=16, gravity=1,

fire\_ability=False, flower\_ability=False)

heroes.append(hero2)

generator = LevelGenerator(platform\_width=32, platform\_height=32,

levels=['data/levels/level1.txt', 'data/levels/level2.txt','data/levels/level3.txt'],

level='', number\_level=0)

'''Start GAME'''

LogicGame.start\_game(settings, generator, background, window, timer, heroes, during\_game=True)

if choice == 3:

''' Move to settings'''

if not during\_game:

LogicMenu.settings(background, window, settings, during\_game)

ViewBackground.fill\_view(background, background\_color)

if choice == 4:

''' Exit Game'''

sys.exit()

@staticmethod

def start\_game(settings, generator, background, window, timer, heroes, during\_game=True):

if generator.number\_level == generator.max\_level:

LogicGame.finish\_game(background, window, 'data/music/lmfao.mp3')

LogicGame.clear\_data(heroes)

else:

entities = pygame.sprite.Group()

for h in heroes.lst:

entities.add(h)

nature = pygame.sprite.Group()

monsters = pygame.sprite.Group()

platforms = []

result\_level = [False]

'''Window Set'''

generator.generate\_level(entities, platforms, monsters, nature)

level\_width = len(generator.level[len(generator.level)-1]) \* generator.platform\_width

level\_height = len(generator.level) \* generator.platform\_height

camera = Camera(level\_width, level\_height, background.get\_width(), background.get\_height())

while True:

button = LogicHero.check\_press\_key(heroes, entities, monsters)

'''Show Menu During Game'''

if button:

choice = LogicMenu.menu(background, window, during\_game=True)

if choice == 1:

''' Move To Settings During Game'''

LogicMenu.settings(background, window, settings, during\_game=True)

heroes.update\_keys(settings)

ViewBackground.fill\_view(background, LogicMenu.background\_color)

if choice == 3:

''' Finish Existing Game'''

if during\_game:

del generator

LogicGame.clear\_data(heroes, entities, monsters, nature)

break

if choice == 4:

''' Exit Game'''

sys.exit()

timer.tick(60)

'''Creating Threads'''

t1 = threading.Thread(target=LogicHero.update\_heroes, args=(heroes, platforms, entities, monsters,

result\_level))

t1.start()

t2 = threading.Thread(target=LogicMonster.update\_monster, args=(monsters, platforms, entities, heroes))

t2.start()

'''Set Background'''

ViewBackground.blit\_view(window, background, 0, 0)

if result\_level[0]:

heroes.set\_default\_settings()

LogicGame.clear\_data(heroes, entities=entities, monsters=monsters, nature=nature, during\_game=True)

LogicGame.start\_game(settings, generator, background, window, timer, heroes, during\_game=True)

result\_level[0] = False

break

'''Check On Existing Heroes'''

if heroes.not\_exist():

break

'''Centering Camera'''

camera.in\_center(heroes.set\_last\_hero().rect)

'''Draw objects'''

for n in nature:

value = camera.shift\_object(n.rect)

ViewBackground.blit\_view(window, n.image, value.x, value.y)

for m in monsters:

if isinstance(m, BlueFlower):

value = camera.shift\_object(m.rect)

ViewBackground.blit\_view(window, m.image, value.x, value.y)

for e in entities:

if not isinstance(e, BlueFlower):

value = camera.shift\_object(e.rect)

ViewBackground.blit\_view(window, e.image, value.x, value.y)

'''Update Background'''

ViewBackground.fill\_view(background, LogicMenu.background\_color)

'''Show lives'''

LogicMenu.show\_interface(background, heroes.get\_lives(), generator.number\_level)

pygame.display.update()

@staticmethod

def clear\_data(heroes, entities=None, monsters=None, nature=None, during\_game=False):

if entities:

for e in entities:

entities.remove(e)

if nature:

for n in nature:

nature.remove(n)

if monsters:

for m in monsters:

monsters.remove(m)

if not during\_game:

heroes.lst.clear()

@staticmethod

def finish\_game(background, window, music\_file\_name):

pygame.mixer.music.load(music\_file\_name)

pygame.mixer.music.play(1, 0)

while True:

time.wait(120)

ViewBackground.fill\_view(background, (92, 146, 252))

ViewBackground.blit\_font(background, pygame.font.Font(LogicMenu.font, random.randint(5,10)\*10),

u'You are WIN', 1, ((random.randint(0, 255), random.randint(0, 255),

(random.randint(0, 255)))), random.randint(0, 500),

random.randint(0, 640))

for e in pygame.event.get():

if e.type == QUIT:

sys.exit()

if e.type == KEYDOWN:

if e.key==K\_ESCAPE:

pygame.mixer.music.stop()

return

ViewBackground.blit\_view(window, background, 0, 0)

pygame.display.flip()

**Приложение В**

**Код класса LogicHero**

import pygame

import sys

from pygame import \*

from model.entity.alive.Fire import Fire

from model.entity.alive.enemies.Bowser import Bowser

from model.entity.alive.enemies.Monster import Monster

from model.entity.alive.enemies.BlueFlower import BlueFlower

from model.entity.alive.enemies.Slub import Slub

from model.entity.alive.heroes.Hero import Hero

from model.entity.dead.BonusBlock import BonusBlock

from model.entity.dead.Flower import Flower

from model.entity.dead.Mushroom import Mushroom

from model.entity.dead.Bonus import Bonus

from model.entity.dead.Exit import Exit

from model.entity.dead.SimpleBlock import SimpleBlock

class LogicHero:

@staticmethod

def check\_press\_key(heroes, entities, monsters):

for e in pygame.event.get():

for hero in heroes.lst:

if e.type == QUIT:

sys.exit()

if e.type == KEYDOWN:

if e.key == hero.key\_fire and hero.fire\_ability:

LogicHero.create\_fire(hero, entities, monsters)

if e.key == hero.key\_left:

hero.left = True

hero.side = True

if e.key == hero.key\_right:

hero.right = True

hero.side = False

if e.key == hero.key\_up:

hero.up = True

if e.key == K\_ESCAPE:

return True

if e.type == KEYUP:

if e.key == hero.key\_right:

hero.right = False

if e.key == hero.key\_left:

hero.left = False

if e.key == hero.key\_up:

hero.up = False

@staticmethod

def update\_heroes(heroes, blocks, entities, monsters, RESULT):

if not heroes.not\_exist():

for block in blocks:

if isinstance(block, BonusBlock):

if block.change\_ability:

block.change\_image()

if isinstance(block, Bonus):

if block.amount\_images\_appearing != 0:

block.appear\_bonus()

else:

block.exist\_bonus()

for hero in heroes.lst:

hero.update()

hero.rect.y += hero.yvel

if hero.rect.y >= 640:

hero.lifes -= 1

heroes.killed(hero, entities)

hero.move\_to\_start()

break

status\_hero = LogicHero.contact\_with\_blocks(heroes, hero, 0, hero.yvel, blocks, entities, monsters)

if status\_hero == True:

RESULT[0] = True

return

hero.rect.x += hero.xvel

status\_hero = LogicHero.contact\_with\_blocks(heroes, hero, hero.xvel, 0, blocks, entities, monsters)

if status\_hero == True:

RESULT[0] = True

return

@staticmethod

def contact\_with\_blocks(heroes, hero, xvel, yvel, blocks, entities, monsters):

if not heroes.not\_exist():

for block in blocks:

hero.update\_bonus()

if sprite.collide\_rect(hero, block):

if xvel > 0:

hero.rect.right = block.rect.left

if isinstance(block, Monster) and block.alive:

hero.lifes -= 1

heroes.killed(hero, entities)

return False

if isinstance(block, Bonus):

LogicHero.contact\_bonus(hero, block, blocks, entities)

LogicHero.contact\_with\_blocks(heroes, hero, xvel, yvel, blocks, entities, monsters)

break

if xvel < 0:

hero.rect.left = block.rect.right

if isinstance(block, Monster) and block.alive:

hero.lifes -= 1

heroes.killed(hero, entities)

return False

if isinstance(block, Bonus):

LogicHero.contact\_bonus(hero, block, blocks, entities)

LogicHero.contact\_with\_blocks(heroes, hero, xvel, yvel, blocks, entities, monsters)

break

if yvel > 0:

hero.rect.bottom = block.rect.top

hero.on\_ground = True

hero.yvel = 0

if (isinstance(block, Bowser) or isinstance(block, BlueFlower)) and block.alive:

hero.lifes -= 1

heroes.killed(hero, entities)

return False

if isinstance(block, Slub) and block.alive:

block.alive = False

block.lifes -=hero.power

block.killed(monsters, entities, blocks)

LogicHero.contact\_with\_blocks(heroes, hero, xvel, yvel, blocks, entities, monsters)

break

if isinstance(block, Bonus):

LogicHero.contact\_bonus(hero, block, blocks, entities)

LogicHero.contact\_with\_blocks(heroes, hero, xvel, yvel, blocks, entities, monsters)

break

if yvel < 0:

hero.rect.top = block.rect.bottom

hero.yvel = 0

if isinstance(block, Monster) and block.alive:

hero.lifes -= 1

heroes.killed(hero, entities)

return False

if isinstance(block, SimpleBlock):

block.lifes -=hero.power

block.killed(blocks, entities)

LogicHero.contact\_with\_blocks(heroes, hero, xvel, yvel, blocks, entities, monsters)

break

if isinstance(block, BonusBlock) and block.activity:

LogicHero.contact\_bonus\_blocks(blocks,block, entities)

LogicHero.contact\_with\_blocks(heroes, hero, xvel, yvel, blocks, entities, monsters)

break

LogicHero.contact\_bonus(hero, block, blocks, entities)

LogicHero.contact\_with\_blocks(heroes, hero, xvel, yvel, blocks, entities, monsters)

break

if isinstance(block, Exit):

time.wait(200)

return True

return False

@staticmethod

def contact\_bonus(hero, block, blocks, entities):

if isinstance(block, Bonus):

if isinstance(block, Mushroom):

hero.time\_mushroom\_activity = block.time\_activity

hero.fire\_ability = True

if isinstance(block, Flower):

hero.get\_super\_jump(block.flower\_value)

hero.time\_flower\_activity = block.time\_activity

hero.flower\_ability = True

entities.remove(block)

blocks.remove(block)

@staticmethod

def contact\_bonus\_blocks(blocks, block, entities):

if isinstance(block, BonusBlock):

block.make\_simple()

bonus = None

if block.type\_bonus == 1:

bonus = Flower(x=block.x + 3, y=block.y - 59, width=26, height=27,

start\_image='data/flower\_1.png',

images\_appearing=['data/flower\_27.png', 'data/flower\_25.png', 'data/flower\_23.png',

'data/flower\_21.png', 'data/flower\_19.png', 'data/flower\_17.png',

'data/flower\_15.png', 'data/flower\_13.png', 'data/flower\_11.png',

'data/flower\_9.png', 'data/flower\_7.png', 'data/flower\_5.png',

'data/flower\_3.png', 'data/flower\_1.png'],

images\_existing=['data/flower\_exist\_day\_1.png', 'data/flower\_exist\_day\_2.png',

'data/flower\_exist\_day\_3.png', 'data/flower\_exist\_day\_4.png'],

koef=30, time\_activity=200000, flower\_value=1.2, change\_ability=True)

if block.type\_bonus == 2:

bonus = Mushroom(x=block.x + 3, y=block.y - 59, width=26, height=27,

start\_image='data/mushroom\_1.png',

images\_appearing=['data/mushroom\_27.png', 'data/mushroom\_25.png', 'data/mushroom\_23.png',

'data/mushroom\_21.png', 'data/mushroom\_19.png', 'data/mushroom\_17.png',

'data/mushroom\_15.png', 'data/mushroom\_13.png', 'data/mushroom\_11.png',

'data/mushroom\_9.png', 'data/mushroom\_7.png', 'data/mushroom\_5.png',

'data/mushroom\_3.png', 'data/mushroom\_1.png'],

images\_existing=['data/mushroom\_exist\_day\_1.png',

'data/mushroom\_exist\_day\_2.png'],

koef=30, time\_activity=200000, change\_ability=True)

if bonus:

blocks.append(bonus)

entities.add(bonus)

@staticmethod

def create\_fire(hero, entities, monsters):

if isinstance(hero, Hero):

if hero.side:

fireball = Fire(x=hero.rect.x-10, y=hero.rect.y+(hero.height//2), width=7, height=6,

image='data/fireball.png', side=hero.side, power=1, xvel=0, yvel=0, gravity=1,

move\_speed=5, left=True, right=False, up=False, on\_ground=False, max\_way=96, alive=True)

else:

fireball = Fire(x=hero.rect.x+hero.width, y=hero.rect.y + (hero.height // 2), width=7, height=6,

image='data/fireball.png', side=hero.side, power=1, xvel=0, yvel=0, gravity=1,

move\_speed=5, left=True, right=False, up=False, on\_ground=False, max\_way=96, alive=True)

entities.add(fireball)

monsters.add(fireball)

**Приложение Г**

**Код класса LogicMenu**

import pygame,sys

from pygame import \*

from view.ViewBackgound import ViewBackground

class LogicMenu:

font = ''

height\_game = 0

height\_menu = 0

background\_color = (0, 0, 0)

color\_active\_text = (0, 0, 0)

color\_simple\_text = (0, 0, 0)

@staticmethod

def show\_interface(background, heroes\_lives, level):

font\_menu = pygame.font.Font(LogicMenu.font, LogicMenu.height\_game)

ViewBackground.blit\_font(background, font\_menu, u'Level %d' % (level), 1, (0, 0, 0), 10, 10)

for i in range(len(heroes\_lives)):

ViewBackground.blit\_font(background, font\_menu, heroes\_lives[i], 1, (0, 0, 0),

background.get\_width()//2+200\*(i-1), 10)

@staticmethod

def get\_points\_menu():

return [(270, 200, u'1 Player', 1),

(270, 300, u'2 Players', 2),

(270, 400, u'Settings', 3),

(270, 500, u'Exit', 4),

(200, 100, u'Welcome To Mario!', 0)]

@staticmethod

def menu(background, window, during\_game=False):

font\_menu = pygame.font.Font(LogicMenu.font, LogicMenu.height\_menu)

point = 1

if during\_game:

points = LogicMenu.get\_points\_during\_menu()

else:

points = LogicMenu.get\_points\_menu()

ViewBackground.fill\_view(background, LogicMenu.background\_color)

while True:

LogicMenu.render(points, background, font\_menu, point)

for e in pygame.event.get():

if e.type == QUIT:

sys.exit()

if e.type == KEYDOWN:

if e.key == K\_ESCAPE:

return

if e.key == K\_UP:

if point > 1:

point -= 1

if e.key == K\_DOWN:

if point < 4:

point += 1

if e.key == 13:

if point in [1, 2, 3, 4]:

return point

ViewBackground.blit\_view(window, background, 0, 0)

pygame.display.flip()

@staticmethod

def get\_settings\_points\_menu(settings):

values=[]

keys = {'space': K\_SPACE, 'tab': K\_TAB, 'right ctrl': K\_RCTRL, 'left ctrl': K\_LCTRL, 'up': K\_UP, 'down': K\_DOWN,

'left': K\_LEFT, 'right': K\_RIGHT, 'delete': K\_DELETE, 'backspace': K\_BACKSPACE, 'f1': K\_F1, 'f2': K\_F2,

'f3': K\_F3, 'f4': K\_F4, 'f5': K\_F5, 'f6': K\_F6, 'f7': K\_F7, 'f8': K\_F8, 'f9': K\_F9, 'f10': K\_F10,

'f11': K\_F11, 'f12': K\_F12, 'f13': K\_F13, 'f14': K\_F14, 'caps lock': K\_CAPSLOCK, 'right shift': K\_RSHIFT,

'left shift': K\_LSHIFT, 'left alt': K\_LALT, 'right alt': K\_RALT, 'menu': K\_MENU, 'insert': K\_INSERT,

'print screen': K\_PRINT, 'end': K\_END, 'page up': K\_PAGEUP, 'page down': K\_PAGEDOWN, 'pause': K\_PAUSE,

'[0]': K\_KP0, '[1]': K\_KP1, '[2]': K\_KP2, '[3]': K\_KP3, '[4]': K\_KP4, '[5]': K\_KP5, '[6]': K\_KP6,

'[7]': K\_KP7, '[8]': K\_KP8, '[9]': K\_KP9, 'enter': K\_KP\_ENTER, '[/]': K\_KP\_DIVIDE, '[+]': K\_KP\_PLUS,

'[-]': K\_KP\_MINUS, '[\*]': K\_KP\_MULTIPLY, '[.]': K\_KP\_PERIOD, 'numlock': K\_NUMLOCK} items = keys.items()

for s in settings:

for k, v in items:

if v == s:

values.append(k)

break

else:

values.append(chr(s))

return [[50, 380, u'Back', 1], [250, 100, u'%s' % values[0], 2], [250, 170, u'%s' % values[1], 3],

[250, 240, u'%s' % values[2], 4], [250, 310, u'%s' % values[3], 5], [550, 100, u'%s' % values[4], 6],

[550, 170, u'%s' % values[5], 7], [550, 240, u'%s' % values[6], 8], [550, 310, u'%s' % values[7], 9],

[50, 100, u'Up', 0], [50, 170, u'Left', 0], [50, 240, u'Right', 0], [50, 310, u'Fire', 0],

[250, 30, u'1 Player', 0], [550, 30, u'2 Player', 0], [50, 100, u'Up', 0], [50, 170, u'Left', 0],

[50, 240, u'Right', 0], [50, 490, u'To change button press Enter', 0],

[50, 540, u'To install selected button', 0], [50, 580, u'press Escape', 0]]

@staticmethod

def settings(background, window, settings, during\_game):

font\_menu = pygame.font.Font(LogicMenu.font, LogicMenu.height\_menu)

point = 1

points = LogicMenu.get\_settings\_points\_menu(settings)

ViewBackground.fill\_view(background, LogicMenu.background\_color)

while True:

LogicMenu.render(points, background, font\_menu, point)

for e in pygame.event.get():

if e.type == QUIT:

sys.exit()

if e.type == KEYDOWN:

if e.key == K\_ESCAPE:

if during\_game:

LogicMenu.menu(background, window, during\_game=True)

return

LogicMenu.menu(background, window, during\_game=False)

return

if e.key == K\_UP:

if point > 1:

point -= 1

if e.key == K\_DOWN:

if point < 9:

point += 1

if point in [2, 3, 4, 5] and e.key == K\_RIGHT:

point += 4

if point in [6, 7, 8, 9] and e.key == K\_LEFT:

point -= 4

if e.key == 13:

if point in [2, 3, 4, 5, 6, 7, 8, 9]:

LogicMenu.replace\_button(background, window, points, point, settings)

if point == 1:

if during\_game:

LogicMenu.menu(background, window, during\_game=True)

return

LogicMenu.menu(background, window, during\_game=False)

return

ViewBackground.blit\_view(window, background, 0, 0)

pygame.display.flip()

@staticmethod

def get\_points\_during\_menu():

return [(250, 200, u'Settings', 1),

(250, 300, u'Back to Game', 2),

(250, 400, u'Finish Game', 3),

(250, 500, u'Exit', 4),

(330, 100, u'Menu', 0)]

@staticmethod

def render(punkts, surface, font, num\_point):

for punkt in punkts:

x, y, text, point = punkt

if num\_point == point and num\_point != 0:

ViewBackground.blit\_font(surface, font, text, 1, LogicMenu.color\_active\_text, x, y)

else:

ViewBackground.blit\_font(surface, font, text, 1, LogicMenu.color\_simple\_text, x, y)

@staticmethod

def replace\_button(background, window, points, point, settings):

font\_menu = pygame.font.Font(LogicMenu.font, LogicMenu.height\_menu)

while True:

LogicMenu.render(points, background, font\_menu, point)

for e in pygame.event.get():

if e.type == QUIT:

sys.exit()

if e.type == KEYDOWN:

if e.key == K\_ESCAPE:

return

else:

for s in settings:

if s == e.key:

break

else:

settings[point - 2] = e.key

points[point - 1][2] = u'%s' % pygame.key.name(e.key)

ViewBackground.fill\_view(background, LogicMenu.background\_color)

ViewBackground.blit\_view(window, background, 0, 0)

pygame.display.flip()

**Приложение Д**

**Код класса LogicMonster**

from pygame import sprite

from model.entity.alive.Fire import Fire

from model.entity.alive.enemies.Bowser import Bowser

from model.entity.alive.enemies.Monster import Monster

class LogicMonster:

@staticmethod

def update\_monster(monsters, blocks, entities, heroes):

for monster in monsters:

if monster.alive:

if isinstance(monster, Monster):

monster.change\_image()

if monster.up:

if monster.on\_ground and not isinstance(monster, Fire):

monster.rect.x = monster.rect.left

monster.yvel = -monster.gravity

if isinstance(monster, Fire):

if monster.side:

monster.rect.x -= monster.move\_speed

else:

monster.rect.x += monster.move\_speed

if monster.left and not isinstance(monster, Fire):

monster.xvel = -monster.move\_speed

if monster.right and not isinstance(monster, Fire):

monster.xvel = monster.move\_speed

if not monster.on\_ground and not isinstance(monster, Fire):

monster.yvel += monster.gravity

monster.on\_ground = False

if not isinstance(monster, Fire):

monster.rect.y += monster.yvel

if LogicMonster.contact\_with\_blocks(monster, monsters, 0, monster.yvel, blocks, entities, heroes) != False:

if not isinstance(monster, Fire):

monster.rect.x += monster.xvel

LogicMonster.contact\_with\_blocks(monster, monsters, monster.xvel, 0, blocks, entities, heroes)

if abs(monster.x-monster.rect.x) > 400:

if isinstance(monster, Fire):

monsters.remove(monster)

entities.remove(monster)

blocks.pop(len(blocks)-1)

if isinstance(monster, Bowser):

monster.side = not(monster.side)

monster.left, monster.right = monster.right, monster.left

@staticmethod

def contact\_with\_blocks(monster, monsters, xvel, yvel, blocks, entities, heroes):

if not heroes.not\_exist():

for e in entities:

if sprite.collide\_rect(monster, e):

if isinstance(monster, Fire) and isinstance(e, Monster):

if isinstance(e, Bowser):

if monster.side != e.side:

e.lifes -= monster.power

e.killed(monsters, entities, blocks)

else:

e.lifes -= monster.power

e.killed(monsters, entities, blocks)

monster.killed(monsters, entities)

return False

if isinstance(monster, Fire) and e.rect.top < monster.rect.top < e.rect.bottom:

monster.killed(monsters, entities)

return False

for block in blocks:

if sprite.collide\_rect(monster, block) and not isinstance(block, Monster):

if xvel > 0:

monster.rect.right = block.rect.left

monster.left = True

monster.right = False

if isinstance(monster, Bowser):

monster.side = not(monster.side)

if xvel < 0:

monster.rect.left = block.rect.right

monster.left = False

monster.right = True

if isinstance(monster, Bowser):

monster.side = not(monster.side)

if yvel > 0:

monster.rect.bottom = block.rect.top

monster.yvel = 0

monster.on\_ground = True

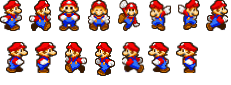
if yvel < 0:

monster.rect.top = block.rect.bottom

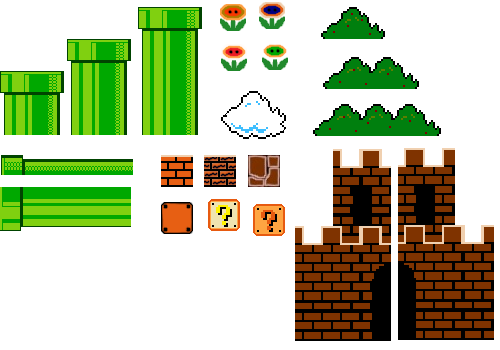
monster.yvel = 0

**Приложение Е**

**Графический материал**

****

****

****

**Приложение Ж**

**Отображение уровня**

