МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
Кафедра компьютерной инженерии и моделирования

РАЗРАБОТКА ИГРОВОГО ПРИЛОЖЕНИЯ ТРЕНИРОВКИ ПАМЯТИ «Саймон говорит»

Курсовая работа
по дисциплине «Программирование»
студента 1 курса группы ПИ-б-о-203
Польского Дениса Александровича

направления подготовки 09.03.04 «Программная инженерия»

Научный руководитель _____
(оценка)
старший преподаватель кафедры
компьютерной инженерии и моделирования _____ Тимофеева С.В.
(подпись, дата)

Симферополь, 2021

РЕФЕРАТ

Польский Д.А. Создание игры на платформе ПК для улучшения своих навыков программирования // Курсовая работа по специальности 09.03.04 Программная инженерия / Кафедра компьютерной инженерии и моделирования Физико-технического института Крымского федерального университета им. В. И. Вернадского. – Симферополь, 2021.

Среда разработки- PyCharm Community Edition 2020.

Цель работы: Улучшить навыки программирования, изучить язык Python, создав игру на ПК.

В современном обществе у большинства людей наблюдаются проблемы с памятью и вниманием. Для предотвращения таких проблем существуют приложения, развивающие память, тренирующие концентрацию внимания, и т.п.

Темой работы выбрал разработку приложения для памяти “Саймон говорит”.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В современном обществе у большинства людей наблюдаются проблемы с памятью и вниманием. Для предотвращения таких проблем существуют приложения, развивающие память, тренирующие концентрацию внимания, и т.п.

Память человека неразрывно связана с мозгом и мыслительными процессами. Чем больше информации мы удерживаем в памяти, тем больше активности у мозга, которому приходится эту информацию обрабатывать. Так мозг обучается и становится более развитым.

Однако уровень интеллекта не фиксируется раз и навсегда в одной точке. Ученые из института Макса Планка в Берлине выяснили, что мозг достигает пика активности в возрасте 16–25 лет, после чего когнитивные функции постепенно снижаются. Но это не значит, что нельзя вернуть мозгу былую силу. Как раз для этого необходимо тренировать мозг и память.

Наш мозг обладает нейропластичностью – способностью адаптироваться к новым условиям и изменяться. И делать это он может в любом возрасте. А лучше всего мозг формирует новые нейронные связи, когда узнает что-то новое и запоминает эту информацию. Также исследования показывают, что регулярные тренировки памяти увеличивают показатели интеллекта у взрослых людей и способны улучшить память пожилых людей и вернуть ее к показателям среднего возраста.

Темой курсового проекта выбрана разработка игрового приложения «Саймон говорит».

В процессе разработки приложения планируется получить навыки, позволяющие реализовывать приложения.

Для упрощения реализации приложения использована библиотека «pygame» предназначенная для Python.

Python — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным — всё является объектами. Необычной особенностью языка является выделение блоков кода пробельными отступами. Синтаксис ядра языка минималистичен, за счёт чего на практике редко возникает необходимость обращаться к документации. Сам же язык известен как интерпретируемый и используется в том числе для написания скриптов. Недостатками языка являются зачастую более низкая скорость работы и более высокое потребление памяти написанных на нём программ по сравнению с аналогичным кодом, написанным на компилируемых языках, таких как Си или C++.

Python является мультипарадигмальным языком программирования, поддерживающим императивное, процедурное, структурное, объектно-ориентированное программирование, метапрограммирование и функциональное программирование. Задачи обобщённого программирования решаются за счёт динамической типизации. Аспектно-ориентированное программирование частично поддерживается через декораторы, более полноценная поддержка обеспечивается дополнительными фреймворками. Такие методики как контрактное и логическое программирование можно реализовать с помощью библиотек или расширений. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений с глобальной блокировкой интерпретатора высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

Целью данного проекта является реализация игрового приложения, тренирующего память «Саймон говорит».

ГЛАВА 1
ПОСТАНОВКА ЗАДАЧИ

1.1 Цель проекта

Основной целью проекта является улучшение навыков программирования. Помимо этого, реализовать полезное приложение, для будущего использования.

1.2 Существующие аналоги

Power Brain Trainer

Приложение, состоящее из нескольких мини игр, предназначенных для повышения умственной способности и тренировки концентрации. Основное преимущество данного приложения это количество мини игр.

Судоку

Приложение-головоломка, ключевой особенностью которого является заполнение клеток игрового поля правильными цифрами, избегая пересечений и повторений.

Math Riddles

Приложение, с упором на математические задания, Предназначенные для тренировки умственной способности.

1.3 Основные отличия от аналогов

В отличии от приведенных выше аналогов, «Саймон говорит» представляет собой минималистическое приложение, способное развивать не только концентрацию внимания, но и так называемую кратковременную память.

1.4 Техническое задание

Разработать приложение, состоящее из 4-х плиток, которые в случайной последовательности загораются, на которые пользователю необходимо нажать в правильной последовательности.

Добавить подсчет очков за одну сессию для отслеживания результата от использования приложения.

Добавить сервер, посылающий результаты пользователя на хранение.

Разместить приложение в play market для мобильных устройств

ГЛАВА 2
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

2.1 Анализ инструментальных средств

Для разработки приложения был выбран модуль Pygame.

Pygame — набор модулей (библиотек) языка программирования Python, предназначенный для написания компьютерных игр и мультимедиа-приложений. Pygame базируется на мультимедийной библиотеке SDL.

Изначально Pygame был написан Питом Шиннерсом (Pete Shinners). Начиная примерно с 2004/2005 года поддерживается и развивается сообществом свободного программного обеспечения.

Pygame – это библиотека модулей для языка Python, созданная для разработки 2D игр. Также Pygame могут называть фреймворком. В программировании понятия "библиотека" и "фреймворк" несколько разные. Но когда дело касается классификации конкретного инструмента, не все так однозначно.

В любом случае, фреймворк является более мощным по-сравнению с библиотекой, он накладывает свою специфику на особенности программирования и сферу использования продукта. С точки зрения специфики Pygame – это фреймворк. Однако его сложно назвать "мощным инструментом". По своему объему и функционалу это скорее библиотека.

Также существует понятие "игрового движка" как программной среды для разработки игр. По своему назначению Pygame можно считать игровым движком. В то же время, с точки зрения классификации программного обеспечения, Pygame является API для Питона к API библиотеки SDL.

API – это интерфейс (в основном набор функций и классов) для прикладного (часто более высокоуровневого) программирования, который предоставляет, например, та или иная библиотека. SDL – это библиотека, которая работает с мультимедийными устройствами компьютера.

2.2 Описание алгоритмов

Алгори́тм — конечная совокупность точно заданных правил решения некоторого класса задач или набор инструкций, описывающих порядок действий исполнителя для решения определённой задачи. В старой трактовке вместо слова «порядок» использовалось слово «последовательность», но по мере развития параллельности в работе компьютеров слово «последовательность» стали заменять более общим словом «порядок». Независимые инструкции могут выполняться в произвольном порядке, параллельно, если это позволяют используемые исполнители. Часто в качестве исполнителя выступает компьютер, но понятие алгоритма необязательно относится к компьютерным программам, так, например, чётко описанный рецепт приготовления блюда также является алгоритмом, в таком случае исполнителем является человек (а может быть и некоторый механизм, ткацкий станок, и пр.).

Началом реализации выступает создание игрового интерфейса приложения. Для этого необходимы параметры. (Рисунок 2.2.2.1)

```
4   FPS = 30
5   WINDOWWIDTH = 640
6   WINDOWHEIGHT = 480
7   FLASHSPEED = 500 # в милисекундах
8   FLASHDELAY = 200 # в милисекундах
9   BUTTONSIZE = 200#
10  BUTTONGAPSIZE = 20
11  TIMEOUT = 4 # секунды, по истичении которых засчитывается проигрыш
12
13  #               R    G    B
14  WHITE       = (255, 255, 255)
15  BLACK       = ( 0,   0,   0)
16  BRIGHTRED   = (255,   0,   0)
17  RED         = (155,   0,   0)
18  BRIGHTGREEN = ( 0, 255,   0)
19  GREEN       = ( 0, 155,   0)
20  BRIGHTBLUE  = ( 0,   0, 255)
21  BLUE        = ( 0,   0, 155)
22  BRIGHTYELLOW = (255, 255,   0)
23  YELLOW      = (155, 155,   0)
24  DARKGRAY    = ( 40,  40,  40)
25  bgColor = BLACK
26
27  XMARGIN = int((WINDOWWIDTH - (2 * BUTTONSIZE) - BUTTONGAPSIZE) / 2)
28  YMARGIN = int((WINDOWHEIGHT - (2 * BUTTONSIZE) - BUTTONGAPSIZE) / 2)
29
```

Рисунок 2.2.2.1. Фрагмент кода с использованными для интерфейса параметрами

Для реализации необходимо использовать бесконечный цикл символизирующий основной игровой цикл. (Рисунок 2.2.2.2)

```
while True: # основной игровой цикл
    clickedButton = None # нажатая кнопка (ЖЕЛТАЯ, КРАСНАЯ, ЗЕЛЕНАЯ или СИНЯЯ)
    DISPLAYSURF.fill(bgColor)
    drawButtons()

    scoreSurf = BASICFONT.render('Score: ' + str(score), 1, WHITE)
    scoreRect = scoreSurf.get_rect()
    scoreRect.topleft = (WINDOWWIDTH - 100, 10)
    DISPLAYSURF.blit(scoreSurf, scoreRect)

    DISPLAYSURF.blit(infoSurf, infoRect)
```

Рисунок 2.2.2.2. Фрагмент кода с бесконечным циклом

После создания цикла идет создание обработки получаемых данных от нажатий пользователя. (Рисунок 2.2.2.2)

```
    for event in pygame.event.get(): # цикл обработки событий
        if event.type == MOUSEBUTTONUP:
            mousex, mousey = event.pos
            clickedButton = getButtonClicked(mousex, mousey)
        elif event.type == KEYDOWN:
            if event.key == K_q:
                clickedButton = YELLOW
            elif event.key == K_w:
                clickedButton = BLUE
            elif event.key == K_a:
                clickedButton = RED
            elif event.key == K_s:
                clickedButton = GREEN
```

Рисунок 2.2.2.2. Фрагмент кода с реализованными кнопками управления

Одним из основных блоков является сама игра, а именно алгоритмы проверки правильности введенной пользователем комбинации и сравнивание ее с заданной случайным образом. (Рисунок 2.2.2.3 и Рисунок 2.2.2.4)

```
if not waitingForInput:
    # игра по шаблону
    pygame.display.update()
    pygame.time.wait(1000)
    pattern.append(random.choice((YELLOW, BLUE, RED, GREEN)))
    for button in pattern:
        flashButtonAnimation(button)
        pygame.time.wait(FLASHDELAY)
    waitingForInput = True
else:
    # ожидание, пока игрок войдет в кнопки
    if clickedButton and clickedButton == pattern[currentStep]:
        # нажатие правильной кнопку
        flashButtonAnimation(clickedButton)
        currentStep += 1
        lastClickTime = time.time()

        if currentStep == len(pattern):
            # нажатие последней кнопки в шаблоне
            changeBackgroundAnimation()
            score += 1
            waitingForInput = False
            currentStep = 0 # возвращение к первому шагу
```

Рисунок 2.2.2.4. Первый блок фрагмента кода алгоритма игрового процесса

```
        elif (clickedButton and clickedButton != pattern[currentStep]) or (currentStep != 0 and time.time() - TIMEOUT > lastClickTime):
            # нажал неправильную кнопку или истекло время ожидания
            gameOverAnimation()
            # сбросить переменные для новой игры:
            pattern = []
            currentStep = 0
            waitingForInput = False
            score = 0
            pygame.time.wait(1000)
            changeBackgroundAnimation()

        pygame.display.update()
        FPSCLOCK.tick(FPS)


def terminate():
    pygame.quit()
    sys.exit()
```

Рисунок 2.2.2.5. Второй блок фрагмента кода алгоритма игрового процесса

Для идентификации порядка последовательности реализована анимация кнопок (Рисунок 2.2.2.6, Рисунок 2.2.2.7)

```
def flashButtonAnimation(color, animationSpeed=50):
    if color == YELLOW:
        flashColor = BRIGHTYELLOW
        rectangle = YELLOWRECT
    elif color == BLUE:
        flashColor = BRIGHTBLUE
        rectangle = BLUERECT
    elif color == RED:
        flashColor = BRIGHTRED
        rectangle = REDRECT
    elif color == GREEN:
        flashColor = BRIGHTGREEN
        rectangle = GREENRECT

    origSurf = DISPLAYSURF.copy()
    flashSurf = pygame.Surface((BUTTONSIZE, BUTTONSIZE))
    flashSurf = flashSurf.convert_alpha()
    r, g, b = flashColor
    for start, end, step in ((0, 255, 1), (255, 0, -1)): # цикл анимации
        for alpha in range(start, end, animationSpeed * step):
            checkForQuit()
            DISPLAYSURF.blit(origSurf, (0, 0))
            flashSurf.fill((r, g, b, alpha))
            DISPLAYSURF.blit(flashSurf, rectangle.topleft)
            pygame.display.update()
            FPSCLOCK.tick(FPS)
    DISPLAYSURF.blit(origSurf, (0, 0))
```

Рисунок 2.2.2.6. Первый фрагмент кода анимации кнопок

```
def drawButtons():
    pygame.draw.rect(DISPLAYSURF, YELLOW, YELLOWRECT)
    pygame.draw.rect(DISPLAYSURF, BLUE,   BLUERECT)
    pygame.draw.rect(DISPLAYSURF, RED,    REDRECT)
    pygame.draw.rect(DISPLAYSURF, GREEN,  GREENRECT)


def changeBackgroundAnimation(animationSpeed=40):
    global bgColor
    newBgColor = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))

    newBgSurf = pygame.Surface((WINDOWWIDTH, WINDOWHEIGHT))
    newBgSurf = newBgSurf.convert_alpha()
    r, g, b = newBgColor
    for alpha in range(0, 255, animationSpeed): # цикл анимации
        checkForQuit()
        DISPLAYSURF.fill(bgColor)

        newBgSurf.fill((r, g, b, alpha))
        DISPLAYSURF.blit(newBgSurf, (0, 0))

        drawButtons() # перерисовываем кнопки поверх тонировки

        pygame.display.update()
        FPSCLOCK.tick(FPS)
    bgColor = newBgColor
```

Рисунок 2.2.2.7. Второй фрагмент кода анимации кнопок

Анимация кнопок при проигрыше представлена на рисунке 2.2.2.8 и 2.2.2.9.

```
def gameOverAnimation(color=WHITE, animationSpeed=50):
    origSurf = DISPLAYSURF.copy()
    flashSurf = pygame.Surface(DISPLAYSURF.get_size())
    flashSurf = flashSurf.convert_alpha()
    r, g, b = color
    for i in range(3): # делает вспышку 3 раза
        for start, end, step in ((0, 255, 1), (255, 0, -1)):
            # Первая итерация в этом цикле устанавливает следующий цикл for
            # перейти от 0 до 255, второй от 255 до 0.
            for alpha in range(start, end, animationSpeed * step): # цикл анимации
                # альфа означает прозрачность. 255 непрозрачно, 0 невидимо
                checkForQuit()
                flashSurf.fill((r, g, b, alpha))
                DISPLAYSURF.blit(origSurf, (0, 0))
                DISPLAYSURF.blit(flashSurf, (0, 0))
                drawButtons()
                pygame.display.update()
                FPSCLOCK.tick(FPS)
```

Рисунок 2.2.2.8. Фрагмент кода анимации при проигрыше

```
207             for alpha in range(start, end, animationSpeed * step): # цикл анимации
208                 # альфа означает прозрачность. 255 непрозрачно, 0 невидимо
209                 checkForQuit()
210                 flashSurf.fill((r, g, b, alpha))
211                 DISPLAYSURF.blit(origSurf, (0, 0))
212                 DISPLAYSURF.blit(flashSurf, (0, 0))
213                 drawButtons()
214                 pygame.display.update()
215                 FPSCLOCK.tick(FPS)
216
217
218
219 def getButtonClicked(x, y):
220     if YELLOWRECT.collidepoint((x, y)):
221         return YELLOW
222     elif BLUERECT.collidepoint((x, y)):
223         return BLUE
224     elif REDRECT.collidepoint((x, y)):
225         return RED
226     elif GREENRECT.collidepoint((x, y)):
227         return GREEN
228     return None
229
230
231 if __name__ == '__main__':
232     main()
```

Рисунок 2.2.2.9. Фрагмент кода анимации при проигрыше

2.3 Описание структур данных

Структура данных — программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в вычислительной технике. Для добавления, поиска, изменения и удаления данных структура данных предоставляет некоторый набор функций, составляющих её интерфейс. Структуры данных формируются с помощью типов данных, ссылок и операций над ними в выбранном языке программирования.

Различные виды структур данных подходят для различных приложений; некоторые из них имеют узкую специализацию для определённых задач. Например, B-деревья обычно подходят для создания баз данных, в то время как

хеш-таблицы используются повсеместно для создания различного рода словарей, например, для отображения доменных имён в интернет-адресах компьютеров.

В данном проекте реализовано несколько блоков данных:

- Блок параметров пользовательского интерфейса (Рисунок 2.2.3.1)

```
FPS = 30
WINDOWWIDTH = 640
WINDOWHEIGHT = 480
FLASHSPEED = 500 # в милисекундах
FLASHDELAY = 200 # в милисекундах
BUTTONSIZE = 200#
BUTTONGAPSIZE = 20
TIMEOUT = 4 # секунды, по истичении которых засчитывается проигрыш
```

Рисунок 2.2.3.1. Блок параметров интерфейса

- Блок кнопок игрового интерфейса (Рисунок 2.2.3.2)

```
# прямоугольные объекты для 4-х кнопок
YELLOWRECT = pygame.Rect(XMARGIN, YMARGIN, BUTTONSIZE, BUTTONSIZE)
BLUERECT   = pygame.Rect(XMARGIN + BUTTONSIZE + BUTTONGAPSIZE, YMARGIN, BUTTONSIZE, BUTTONSIZE)
REDRECT    = pygame.Rect(XMARGIN, YMARGIN + BUTTONSIZE + BUTTONGAPSIZE, BUTTONSIZE, BUTTONSIZE)
GREENRECT  = pygame.Rect(XMARGIN + BUTTONSIZE + BUTTONGAPSIZE, YMARGIN + BUTTONSIZE + BUTTONGAPSIZE, BUTTONSIZE, BUTTONSIZE)
```

Рисунок 2.2.3.2. Блок кнопок игрового интерфейса

- Блок цветов заднего фона (Рисунок 2.2.3.3)

```
WHITE       = (255, 255, 255)
BLACK       = (  0,   0,   0)
BRIGHTRED   = (255,   0,   0)
RED         = (155,   0,   0)
BRIGHTGREEN = (  0, 255,   0)
GREEN       = (  0, 155,   0)
BRIGHTBLUE  = (  0,   0, 255)
BLUE        = (  0,   0, 155)
BRIGHTYELLOW = (255, 255,   0)
YELLOW      = (155, 155,   0)
DARKGRAY    = ( 40,  40,  40)
bgColor = BLACK
```

Рисунок 2.2.3.3. Блок кодов цветов

- Блок промежуточных переменных (Рисунок.2.2.3.4)

```
# Инициализация некоторых переменных для новой игры
pattern = [] # хранит образец цвета
currentStep = 0 # цвет, который игрок должен выбрать следующим
lastClickTime = 0 # отметка времени последнего нажатия кнопки игроком
score = 0
```

Рисунок 2.2.3.4. Блок промежуточных переменных

2.4 Описание основных модулей

Модуль — функционально законченный фрагмент программы, оформленный в виде отдельного файла с исходным кодом. Модули проектируются таким образом, чтобы предоставлять программистам удобную для многократного использования функциональность (интерфейс) в виде набора функций, классов, констант. Модули могут объединяться в пакеты и, далее, в библиотеки. Удобство использования модульной архитектуры заключается в возможности обновления (замены) модуля, без необходимости изменения остальной системы.

Использование модулей позволяет упростить тестирование программы и обнаружение ошибок. Модульность часто является средством упрощения задачи проектирования программы и распределения процесса разработки между группами разработчиков. При разбиении программы на модули для каждого модуля указывается реализуемая им функциональность, а также связи с другими модулями.

Программа делится на два основных модуля: основной игровой цикл и «тело» игры.

Игровой цикл — принцип, согласно которому геймдизайнеры задают главный элемент игровой механики, который определяет фундаментальный опыт игрока. Один игровой цикл представляет собой действие игрока, результат этого

действия в игровом мире, реакцию игрока на результат и запрос игры на повторение нового действия.

Повторение является одним из фундаментальных аспектов игры. Когда люди получают удовольствие от чего-то, то они хотят это повторить. Например, дети чрезвычайно любят пересматривать мультфильмы снова и снова, или постоянно спрашивают, чтобы им прочитали одну и ту же сказку на ночь. Аналогичный процесс происходит для более старшего возраста и данный аспект переносится на построение игровой механики геймдизайнером. Во время разработки игры основной игровой цикл является центральным блоком, на основании которого строится игра. Обычно он представлен в виде глаголов: выстрелить, прыгнуть, посмотреть состояние и т. п. В идеальном случае игровой цикл может быть описан несколькими словами, которые отражают тот опыт, который получает игрок.

ГЛАВА 3
ТЕСТИРОВАНИЕ ПРОГРАММЫ

3.1 Тестирование исходного кода

В результате тестирование моего кода критических ошибок и недочетов не было обнаружено

3.2 Тестирование интерфейса пользователя и юзабилити

Мой проект прошел тестирование на юзабилити, все тесты были успешно пройдены. Ниже представлены скриншоты тест-кейсов и результаты.

Тест 1/Test Name 1

Среда тестирования /Environment	PyCharm
Предварительные действия /Pre Requisites	Запуск среды разработки PyCharm
Комментарии /Comments	Необходимо установить модуль PyGame. Для это в Terminal необходимо написать: pip install pygame

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Отладка и запуск программы	Программа запускается. Корректно отображаются все кнопки и окна на экране.	Работает как ожидается	Пройден	
2	Нажать на крестик	Окно программы закрывается	Работает как ожидается	Пройден	

Рисунок 3.3.2.1. Первый тест-кейс

Тест 2/Test Name 2

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Нажать на кнопки q,w,a,s	Плитки подсвечиваются	Работает как ожидается	Пройден	
2	Нажать на все плитки левой кнопкой мыши	Плитки подсвечиваются	Работает как ожидается	Пройден	

Рисунок 3.3.2.2. Второй тест-кейс

Тест 3/Test Name 3

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Ввести всю комбинацию правильно	Переход на следующий уровень, с характерным изменением заднего фона	Работает как ожидается	Пройден	

Рисунок 3.3.2.3. Третий тест-кейс

Тест 4/Test Name 4

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Допустить ошибку в правильной комбинации	Экран начинает мерцать и счетчик очков обнуляется	Работает как ожидается	Пройден	

Рисунок 3.3.2.4. Четвертый тест-кейс

ГЛАВА 4
ПЕРСПЕКТИВЫ ДАЛЬНЕЙШЕГО РАЗВИТИЯ ПРОЕКТА

4.1 Перспективы технического развития

Тут перечисляются возможности, которые не были реализованы в проекте, были недоделаны или реализованы не так как изначально задумывалось. Так же указывается, к каким улучшениям может привести добавление этих возможностей в проект.

Основной утратой в реализации приложения является отсутствие сохранения очков за каждую сессию. Так же отсутствие портативности игрового приложения.

После портирования приложения на мобильные устройства, игровое приложение будет пользоваться большим спросом, из-за доступности и простоты использования.

Помимо этого возможным улучшение данного приложения является расширение функционала, а именно добавление новы мини игр, способствующих развитию не только кратковременной памяти но и мышления в целом.

4.2 Перспективы монетизации

В этом разделе перечисляются возможные способы монетизации проекта, как уже реализованные (если такие есть), так и просто идеи.

Так как проект не доработан лучшая перспектива монетизации – это портирование приложения на мобильные устройства и размещение в магазине приложений «Play market» с интеграцией не навязчивой рекламы в сам интерфейс приложения.

ЗАКЛЮЧЕНИЕ

В результате создания данного проекта получено много опыта, выполнена цель по созданию полноценной игры. Самое важное в данном проекте, это то, что он принес большое количество практики программирования и улучшил мои навыки написания кода. Так же мною были получены навыки смежной с программированием сферы разработки игр:

- Навыки геймдизайнера
- Практика в тестировании приложений
- Получен опыт написания документации и отчетности по деятельности

ЛИТЕРАТУРА

1. ГОСТ 19.002-80 Схемы алгоритмов и программ. Правила выполнения [Текст] – Введ. с 01.07. 1981 г. М.: Изд-во стандартов, 1981. – 9 с.

2. ГОСТ 19.003-80 Схемы алгоритмов и программ. Обозначение условные графические [Текст] – Введ. с 01.07. 1981 г. М.: Изд-во стандартов, 1981. – 9 с.

3. Оформление выпускной квалификационной работы на соискание квалификационного уровня «Магистр» («Бакалавр»): методические рекомендации. / сост. Бержанский В.Н., Дзедолик И.В., Полулях С.Н. – Симферополь: КФУ им. В.И.Вернадского, 2017. – 31 с.

4. Документация Pygame [Электронный ресурс] URL:https://www.pygame.org/docs/

ПРИЛОЖЕНИЕ 1
КОД ОСНОВНЫХ МОДУЛЕЙ ПРОЕКТА

```
import random, sys, time, pygame
from pygame.locals import *

FPS = 30
WINDOWWIDTH = 640
WINDOWHEIGHT = 480
FLASHSPEED = 500 # в милисекундах
FLASHDELAY = 200 # в милисекундах
BUTTONSIZE = 200#
BUTTONGAPSIZE = 20
TIMEOUT = 4 # секунды, по истичении которых засчитывается проигрыш

#               R    G    B
WHITE       = (255, 255, 255)
BLACK       = (  0,   0,   0)
BRIGHTRED   = (255,   0,   0)
RED         = (155,   0,   0)
BRIGHTGREEN = (  0, 255,   0)
GREEN       = (  0, 155,   0)
BRIGHTBLUE  = (  0,   0, 255)
BLUE        = (  0,   0, 155)
BRIGHTYELLOW = (255, 255,   0)
YELLOW      = (155, 155,   0)
DARKGRAY    = ( 40,  40,  40)
bgColor = BLACK
```

```
    XMARGIN = int((WINDOWWIDTH - (2 * BUTTONSIZE) - BUTTONGAPSIZE) / 2)
    YMARGIN = int((WINDOWHEIGHT - (2 * BUTTONSIZE) - BUTTONGAPSIZE) / 2)

    # прямоугольные объекты для 4-х кнопок
    YELLOWRECT = pygame.Rect(XMARGIN, YMARGIN, BUTTONSIZE, BUTTONSIZE)
    BLUERECT = pygame.Rect(XMARGIN + BUTTONSIZE + BUTTONGAPSIZE, YMARGIN, BUTTONSIZE, BUTTONSIZE)
    REDRECT = pygame.Rect(XMARGIN, YMARGIN + BUTTONSIZE + BUTTONGAPSIZE, BUTTONSIZE, BUTTONSIZE)
    GREENRECT = pygame.Rect(XMARGIN + BUTTONSIZE + BUTTONGAPSIZE, YMARGIN + BUTTONSIZE + BUTTONGAPSIZE, BUTTONSIZE, BUTTONSIZE)

    def main():
        global FPSCLOCK, DISPLAYSURF, BASICFONT, BEEP1, BEEP2, BEEP3, BEEP4

        pygame.init()
        FPSCLOCK = pygame.time.Clock()
        DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
        pygame.display.set_caption('Саймон говорит')

        BASICFONT = pygame.font.Font('freesansbold.ttf', 16)
        infoSurf = BASICFONT.render('Управление: ЛКМ, клавиши: q,w,a,s', 1, DARKGRAY)
        infoRect = infoSurf.get_rect()
```

```
    infoRect.topleft = (180, WINDOWHEIGHT - 20)



    # Инициализация некоторых переменных для новой игры
    pattern = [] # хранит образец цвета
    currentStep = 0 # цвет, который игрок должен выбрать следующим
    lastClickTime = 0 # отметка времени последнего нажатия кнопки игроком
    score = 0
    # если False, шаблон воспроизводится. когда True, ожидая, пока игрок нажмет цветную кнопку:
    waitingForInput = False

    while True: # основной игровой цикл
      clickedButton = None # нажатая кнопка (ЖЕЛТАЯ, КРАСНАЯ, ЗЕЛЕНАЯ или СИНЯЯ)
      DISPLAYSURF.fill(bgColor)
      drawButtons()

      scoreSurf = BASICFONT.render('Score: ' + str(score), 1, WHITE)
      scoreRect = scoreSurf.get_rect()
      scoreRect.topleft = (WINDOWWIDTH - 100, 10)
      DISPLAYSURF.blit(scoreSurf, scoreRect)

      DISPLAYSURF.blit(infoSurf, infoRect)

      checkForQuit()
      for event in pygame.event.get(): # цикл обработки событий
        if event.type == MOUSEBUTTONUP:
```

```
            mousex, mousey = event.pos
            clickedButton = getButtonClicked(mousex, mousey)
        elif event.type == KEYDOWN:
            if event.key == K_q:
                clickedButton = YELLOW
            elif event.key == K_w:
                clickedButton = BLUE
            elif event.key == K_a:
                clickedButton = RED
            elif event.key == K_s:
                clickedButton = GREEN



    if not waitingForInput:
        # игра по шаблону
        pygame.display.update()
        pygame.time.wait(1000)
        pattern.append(random.choice((YELLOW, BLUE, RED,
GREEN)))
        for button in pattern:
            flashButtonAnimation(button)
            pygame.time.wait(FLASHDELAY)
        waitingForInput = True
    else:
        # ожидаение, пока игрок нажмет на кнопки
        if clickedButton and clickedButton == pattern[currentStep]:
            # нажатие правильной кнопку
            flashButtonAnimation(clickedButton)
            currentStep += 1
```

```
                lastClickTime = time.time()

                if currentStep == len(pattern):
                    # нажатие последней кнопки в шаблоне
                    changeBackgroundAnimation()
                    score += 1
                    waitingForInput = False
                    currentStep = 0 # возвращение к первому шагу

            elif (clickedButton and clickedButton != pattern[currentStep]) or (currentStep != 0 and time.time() - TIMEOUT > lastClickTime):
                # нажал неправильную кнопку или истекло время ожидания
                gameOverAnimation()
                # сбросить переменные для новой игры:
                pattern = []
                currentStep = 0
                waitingForInput = False
                score = 0
                pygame.time.wait(1000)
                changeBackgroundAnimation()

        pygame.display.update()
        FPSCLOCK.tick(FPS)


def terminate():
    pygame.quit()
    sys.exit()
```

```
    def checkForQuit():
        for event in pygame.event.get(QUIT): # получить все события выхода
            terminate() # прекратить, если присутствуют какие-либо события QUIT
        for event in pygame.event.get(KEYUP): # получить все события KEYUP
            if event.key == K_ESCAPE:
                terminate() # прекратить, если событие KEYUP было для клавиши Esc
            pygame.event.post(event) # верните другие объекты события KEYUP


    def flashButtonAnimation(color, animationSpeed=50):
        if color == YELLOW:
            flashColor = BRIGHTYELLOW
            rectangle = YELLOWRECT
        elif color == BLUE:
            flashColor = BRIGHTBLUE
            rectangle = BLUERECT
        elif color == RED:
            flashColor = BRIGHTRED
            rectangle = REDRECT
        elif color == GREEN:
            flashColor = BRIGHTGREEN
            rectangle = GREENRECT

        origSurf = DISPLAYSURF.copy()
        flashSurf = pygame.Surface((BUTTONSIZE, BUTTONSIZE))
```

```
        flashSurf = flashSurf.convert_alpha()
        r, g, b = flashColor
        for start, end, step in ((0, 255, 1), (255, 0, -1)): # цикл анимации
            for alpha in range(start, end, animationSpeed * step):
                checkForQuit()
                DISPLAYSURF.blit(origSurf, (0, 0))
                flashSurf.fill((r, g, b, alpha))
                DISPLAYSURF.blit(flashSurf, rectangle.topleft)
                pygame.display.update()
                FPSCLOCK.tick(FPS)
        DISPLAYSURF.blit(origSurf, (0, 0))


    def drawButtons():
        pygame.draw.rect(DISPLAYSURF, YELLOW, YELLOWRECT)
        pygame.draw.rect(DISPLAYSURF, BLUE,   BLUERECT)
        pygame.draw.rect(DISPLAYSURF, RED,    REDRECT)
        pygame.draw.rect(DISPLAYSURF, GREEN,  GREENRECT)


    def changeBackgroundAnimation(animationSpeed=40):
        global bgColor
        newBgColor = (random.randint(0, 255), random.randint(0, 255),
random.randint(0, 255))

        newBgSurf = pygame.Surface((WINDOWWIDTH,
WINDOWHEIGHT))
        newBgSurf = newBgSurf.convert_alpha()
        r, g, b = newBgColor
        for alpha in range(0, 255, animationSpeed): # цикл анимации
```

```
            checkForQuit()
            DISPLAYSURF.fill(bgColor)

            newBgSurf.fill((r, g, b, alpha))
            DISPLAYSURF.blit(newBgSurf, (0, 0))

            drawButtons() # перерисовываем кнопки поверх тонировки

            pygame.display.update()
            FPSCLOCK.tick(FPS)
        bgColor = newBgColor


    def gameOverAnimation(color=WHITE, animationSpeed=50):
        origSurf = DISPLAYSURF.copy()
        flashSurf = pygame.Surface(DISPLAYSURF.get_size())
        flashSurf = flashSurf.convert_alpha()
        r, g, b = color
        for i in range(3): # делает вспышку 3 раза
            for start, end, step in ((0, 255, 1), (255, 0, -1)):
                # Первая итерация в этом цикле устанавливает следующий
цикл for
                # перейти от 0 до 255, второй от 255 до 0.
                for alpha in range(start, end, animationSpeed * step): # цикл
анимации
                    # альфа означает прозрачность. 255 непрозрачно, 0
невидимо
                    checkForQuit()
                    flashSurf.fill((r, g, b, alpha))
                    DISPLAYSURF.blit(origSurf, (0, 0))
```

```
                DISPLAYSURF.blit(flashSurf, (0, 0))
                drawButtons()
                pygame.display.update()
                FPSCLOCK.tick(FPS)



    def getButtonClicked(x, y):
        if YELLOWRECT.collidepoint( (x, y) ):
            return YELLOW
        elif BLUERECT.collidepoint( (x, y) ):
            return BLUE
        elif REDRECT.collidepoint( (x, y) ):
            return RED
        elif GREENRECT.collidepoint( (x, y) ):
            return GREEN
        return None


    if __name__ == '__main__':
main()
```