Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение высшего образования  
«Рязанский государственный радиотехнический университет имени В.Ф. Уткина»  
Кафедра «Вычислительная и прикладная математика»

**Пояснительная записка  
к курсовой работе по дисциплине**  
«Программирование на языке высокого уровня» **на тему**  
«Игровые программы»

Выполнил: ст. гр. 1413  
Рогатин А. В.  
Проверил: доцент  
Антипов О. В.

Рязань 2022

**Оглавление**

[Задание на курсовую работу 3](#_Toc105689318)

[Введение 4](#_Toc105689319)

[1. Анализ задания и математическая постановка задачи 5](#_Toc105689320)

[2. Разработка схемы алгоритма и её описание 6](#_Toc105689321)

[3. Разработка схемы алгоритма и её описание 19](#_Toc105689322)

[4. Инструкция по использованию разработанной программы 21](#_Toc105689323)

[5. Результаты отладки 24](#_Toc105689324)

[6. Решение контрольных примеров 26](#_Toc105689325)

[7. Календарь выполнения работ 29](#_Toc105689326)

[Заключение 30](#_Toc105689327)

[Список используемой литературы 31](#_Toc105689328)

[Приложение 32](#_Toc105689329)

Задание на курсовую работу

Федеральное агентство по образованию Российской Федерации  
Рязанский государственный радиотехнический университет  
Факультет вычислительной техники  
Кафедра вычислительной и прикладной математики

ЗАДАНИЕ  
на курсовую работу по дисциплине  
«Программирование на языке высокого уровня»  
студенту Рогатину А.В. группы 1413.

Создать игру, происходящую в режиме реального времени, выполненную в псевдо 3d с видом от первого лица. Управление производится перемещением и нажатиями мышью. Игра заключается в том, чтобы продержаться определённое время против врага, с которыми можно бороться, контактируя со специальными кнопками и картой местоположения врагов. Количество нажатий на кнопки ограничено и его нужно производить только в определённое время, чтобы они сработали, поэтому игроку придётся разрабатывать тактику и находить правильный порядок нажатий.

Срок представления работы к защите: 25 мая 2022 года

Дата выдачи задания: 10.02.22

Преподаватель: Антипов О. В.

Введение

Сфера компьютерных развлечений сейчас развивается огромными шагами. За один день в мире создаются тысячи игр, принося игровой индустрии миллионные доходы. Этому игровые разработчики отчасти обязаны игровым движкам. Игровой движок – базовое программное обеспечение компьютерной игры. Уже создано много готовых игровых движков: Unity, Unreal Engine. Они безусловно упрощают работу по созданию игр, но при этом чрезмерный уровень абстракции иногда может препятствовать добавлению сложного и необычного функционала в игру. В данной ситуации приходится менять код самого игрового движка, что может быть достаточно тяжело для неподготовленного разработчика. Поэтому опыт создания игр без игровых движков может дать представление о том, как работают игровые движки, а соответственно поможет работать с игровыми движками более эффективно.

Главной целью будет создание конечного продукта – игры. Из этой цели вытекают следующие задачи:

* Создание удобных функций для прорисовки
* Обработка ввода с клавиатуры и мышки
* Разработка системы условий
* Проработка визуальной части игры

В качестве рабочей среды выбран язык Pascal и редактор Pascal ABC. При этом в процессе разработки используются стандартный модуль GraphWPF.

1. Анализ задания и математическая постановка задачи

В программе будет реализована сама игровая часть, меню, пауза и система сохранений прогресса в ещё незавершённой игре.

Поскольку игра происходит в режиме реального времени, то будут реализованы масштабирование изображений в зависимости от размеров окна и контроль скорости действий в зависимости от количества кадров в секунду.

Игрок контактирует с объектами на экране при помощи мышки. Он может удерживать кнопки, чтобы активировать их. В игровой части пользователь может контактировать с выделенными областями по бокам экрана, чтобы получать доступ к разным действиям и информации. Так же игрок может нажимать на изображения дверей и вентиляции, чтобы отпугивать врагов, прячущихся за ними. Также в игровой части будет реализована пауза по нажатию на клавишу «Esc».

Программа выводит на экран изображения и текст в зависимости от состояния, в котором она находится.

1. Состояние предупреждение

Перед входом в меню появляется оповещение, предупреждающее о возможности наличия в игре громких звуков и

1. Состояние меню

В состоянии меню игрок видит перед собой кнопки начала игры, продолжения игры и выхода.

1. Состояние оповещение перед стартом игры

Перед стартом игры появляется оповещение, рассказывающее предысторию действий игры.

1. Состояние игра

Данное состояние делится на подсостояния, в зависимости от которых на экран выводятся разные изображения.

1. Состояние пауза

В состоянии пауза игрок видит надпись «Пауза» и кнопку сохранения с выходом из программы

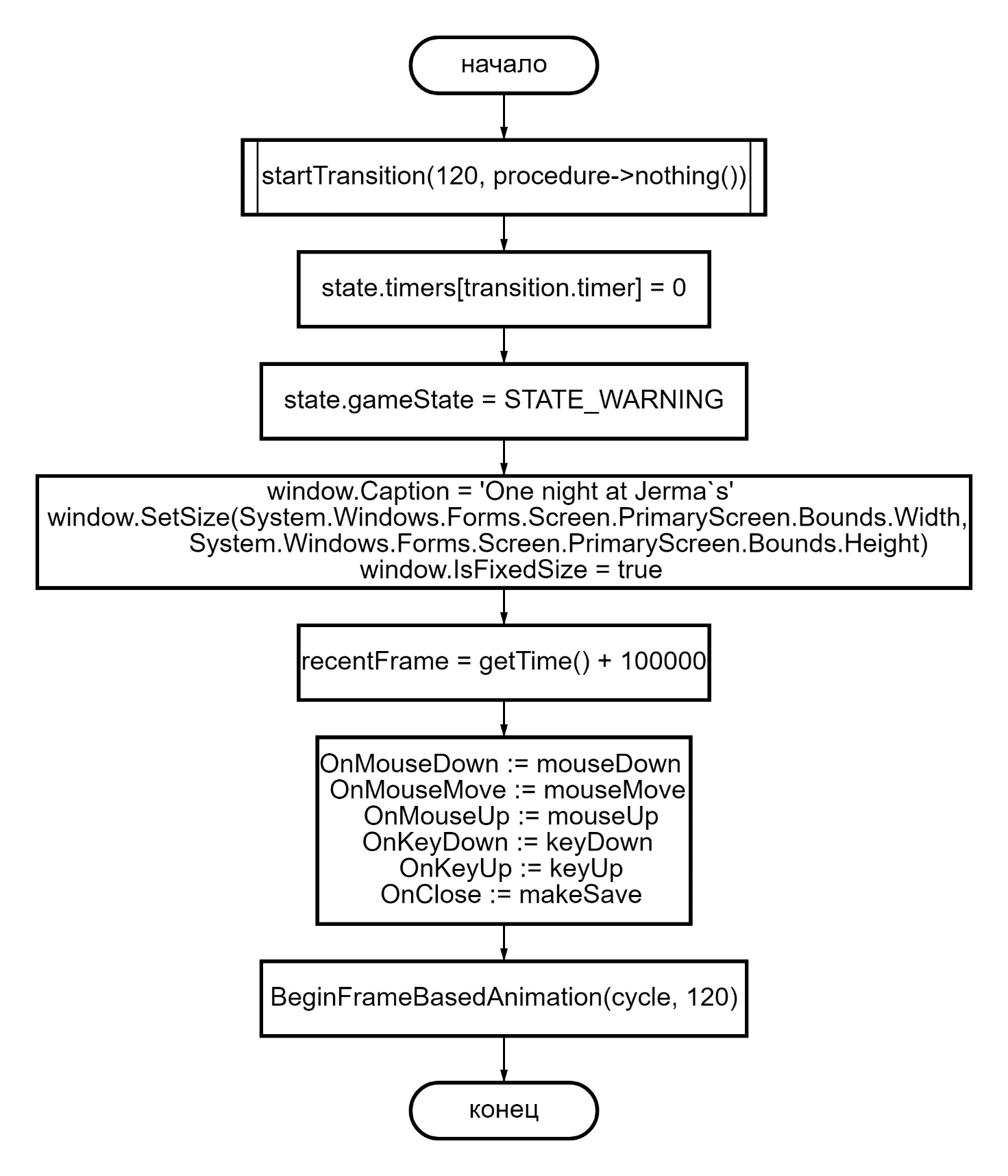
1. Состояние поражение

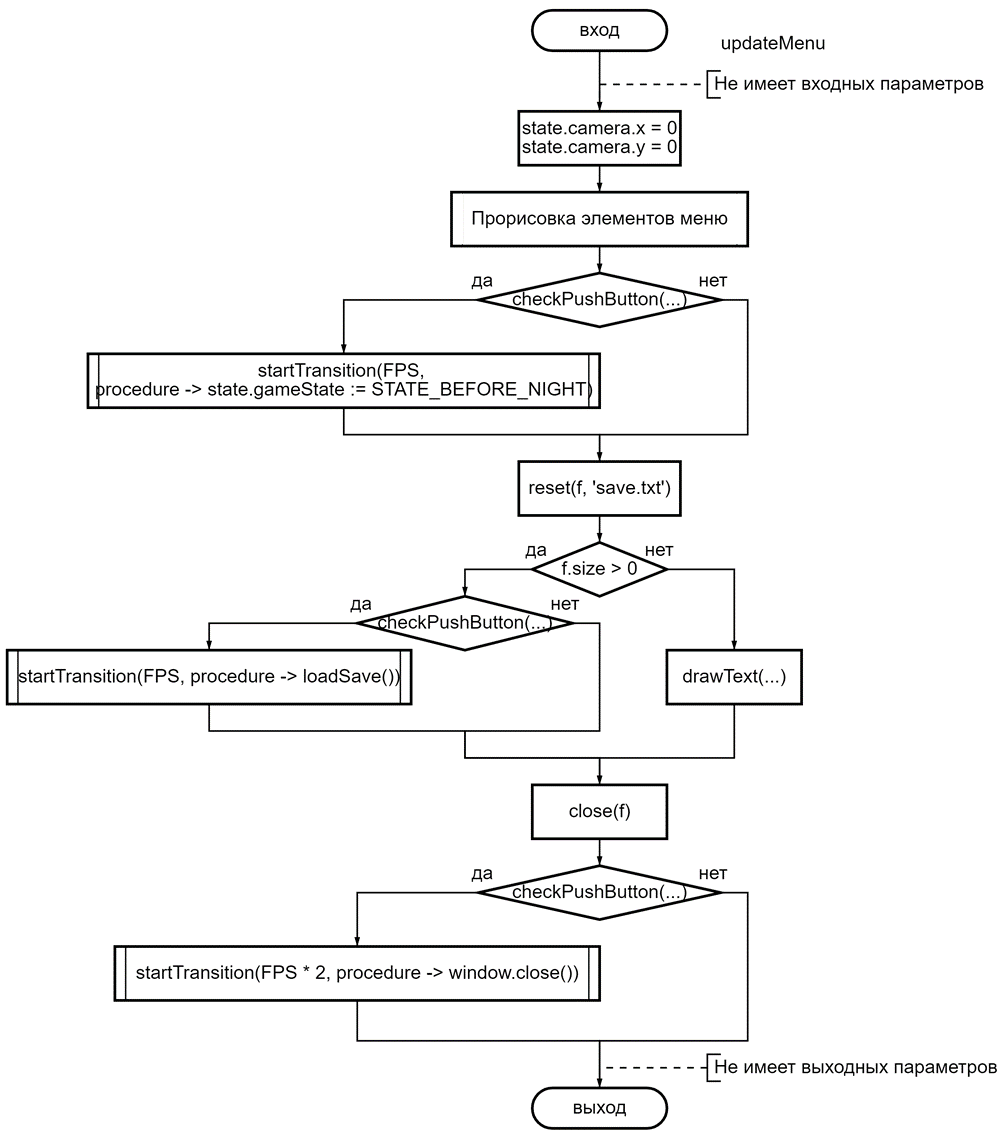
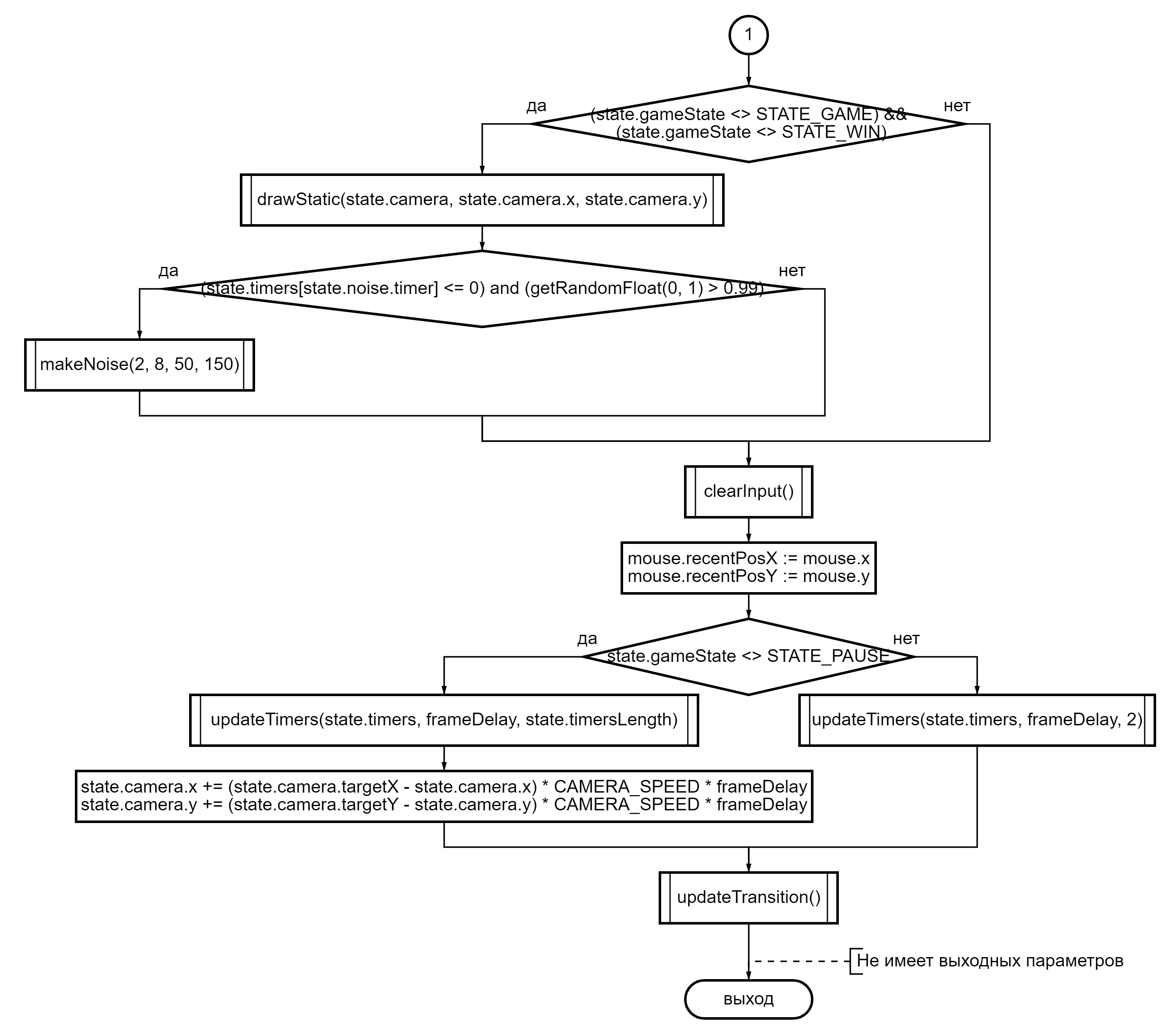
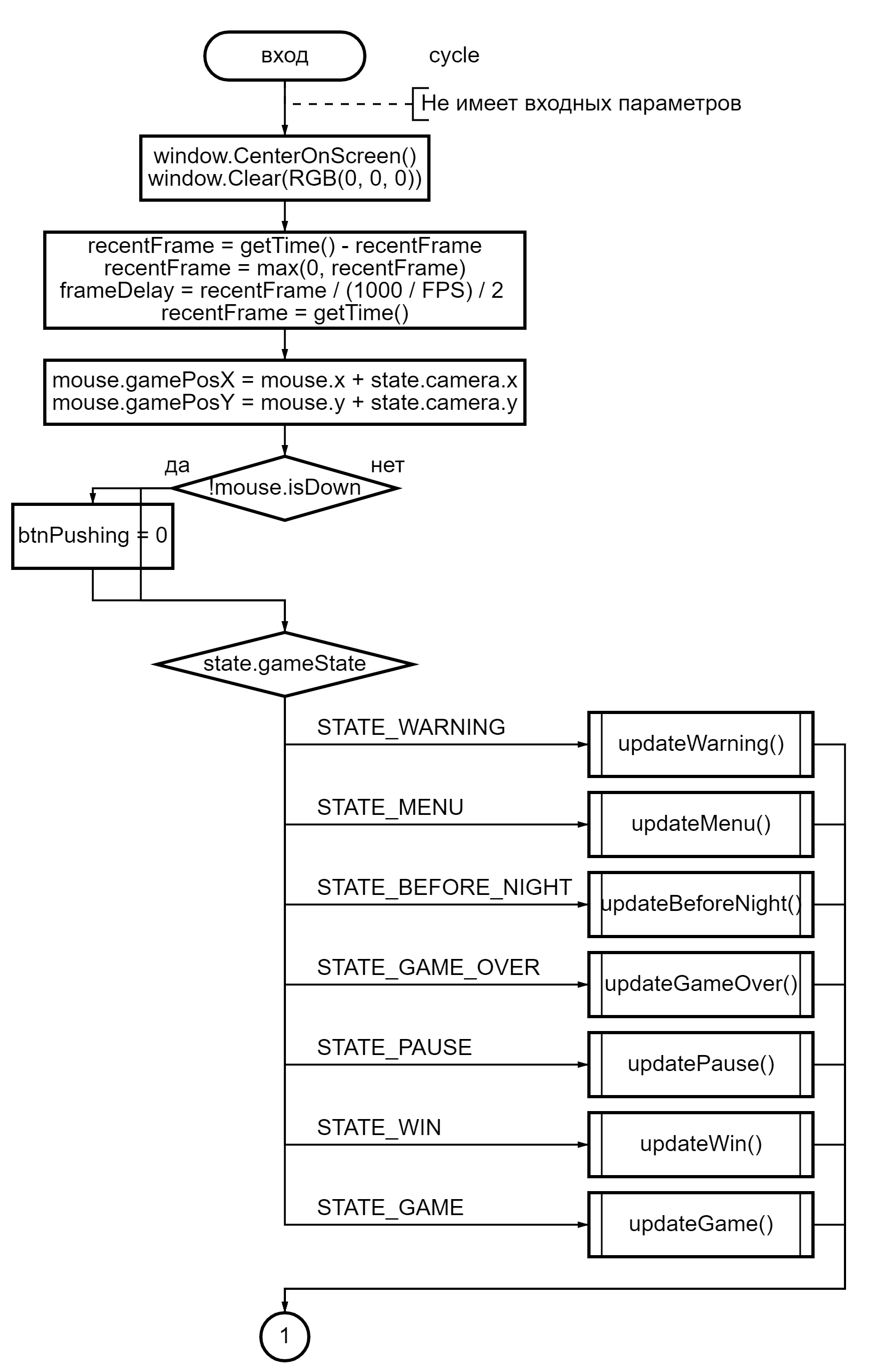
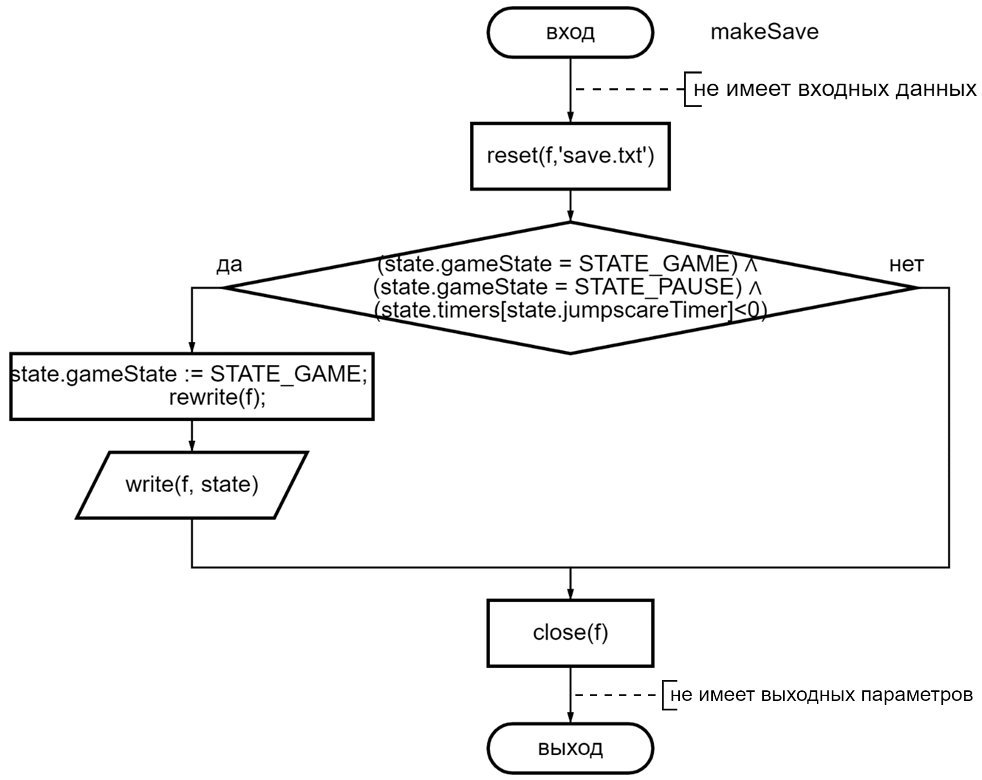
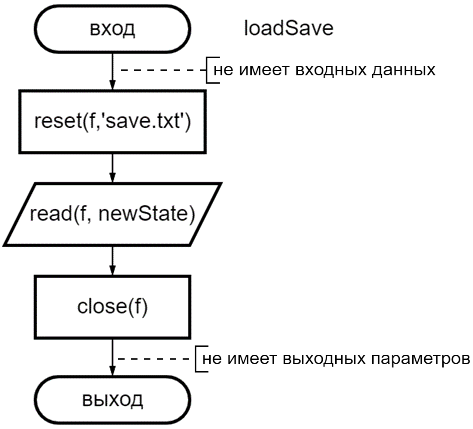
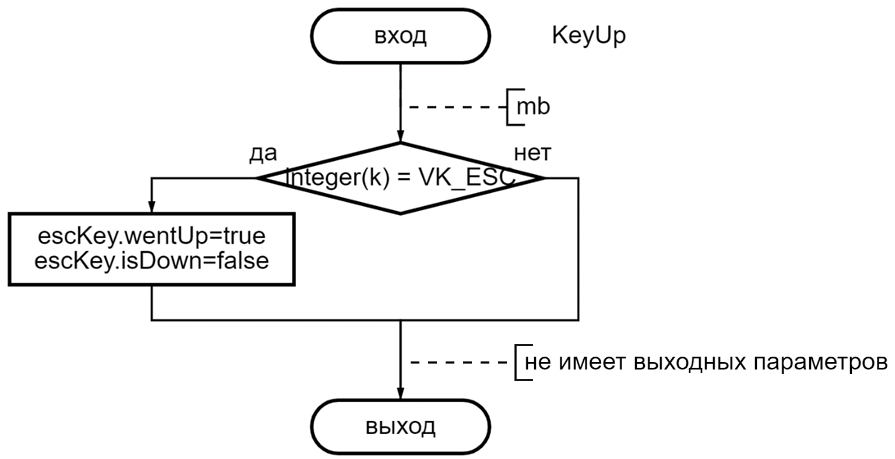
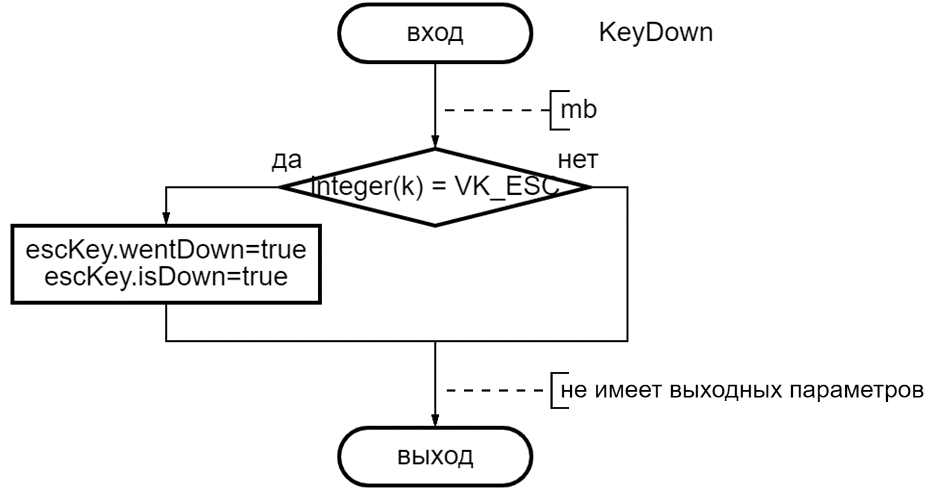
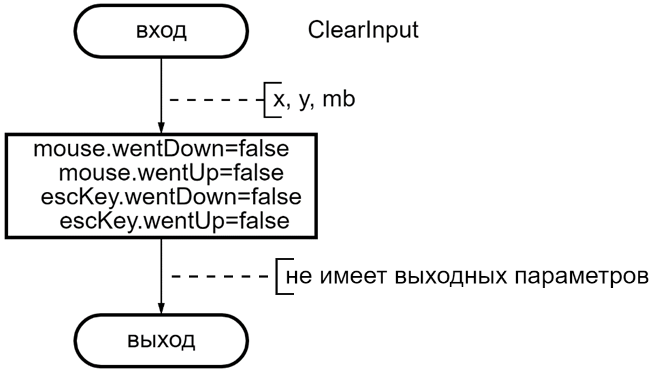
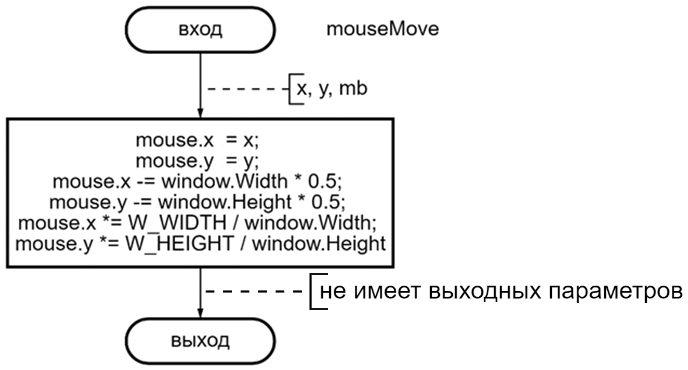
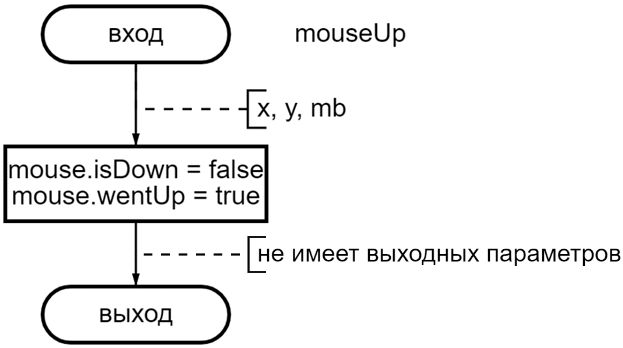
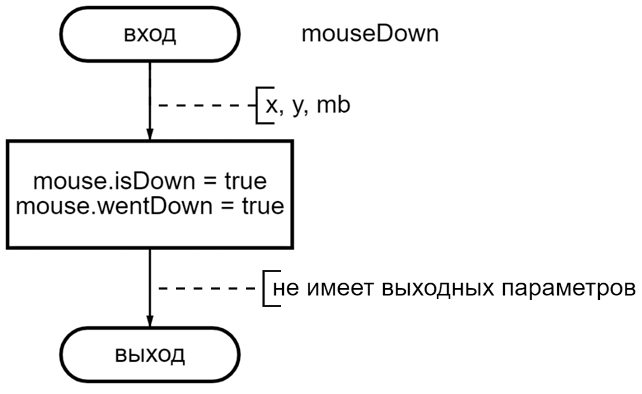
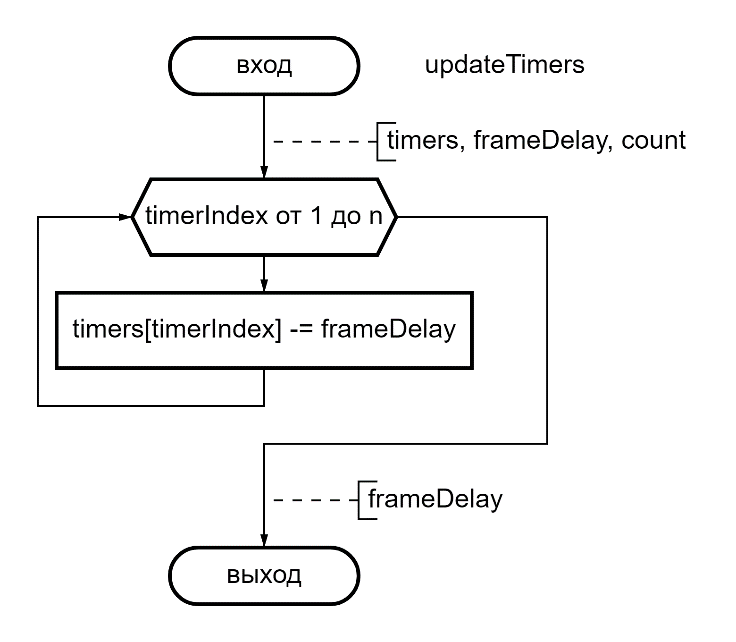
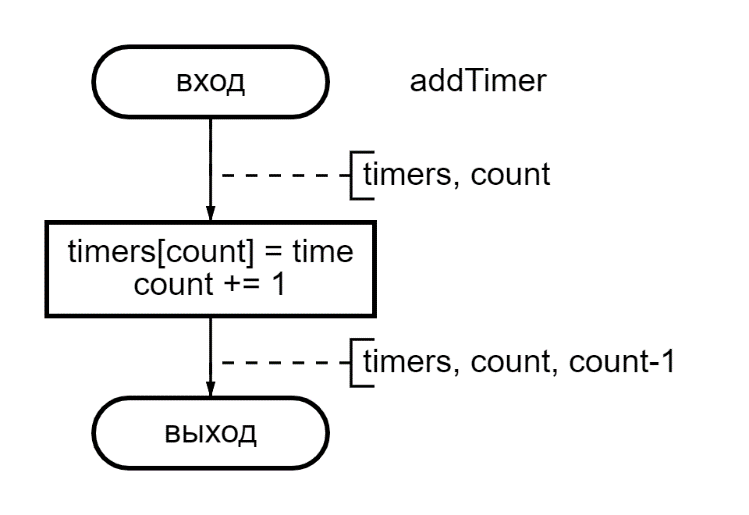
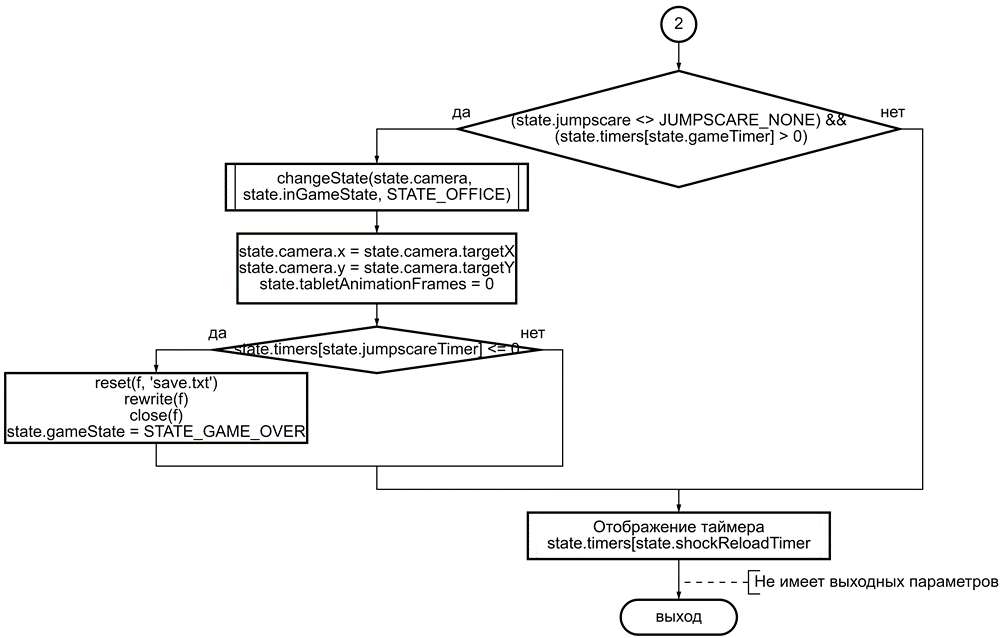
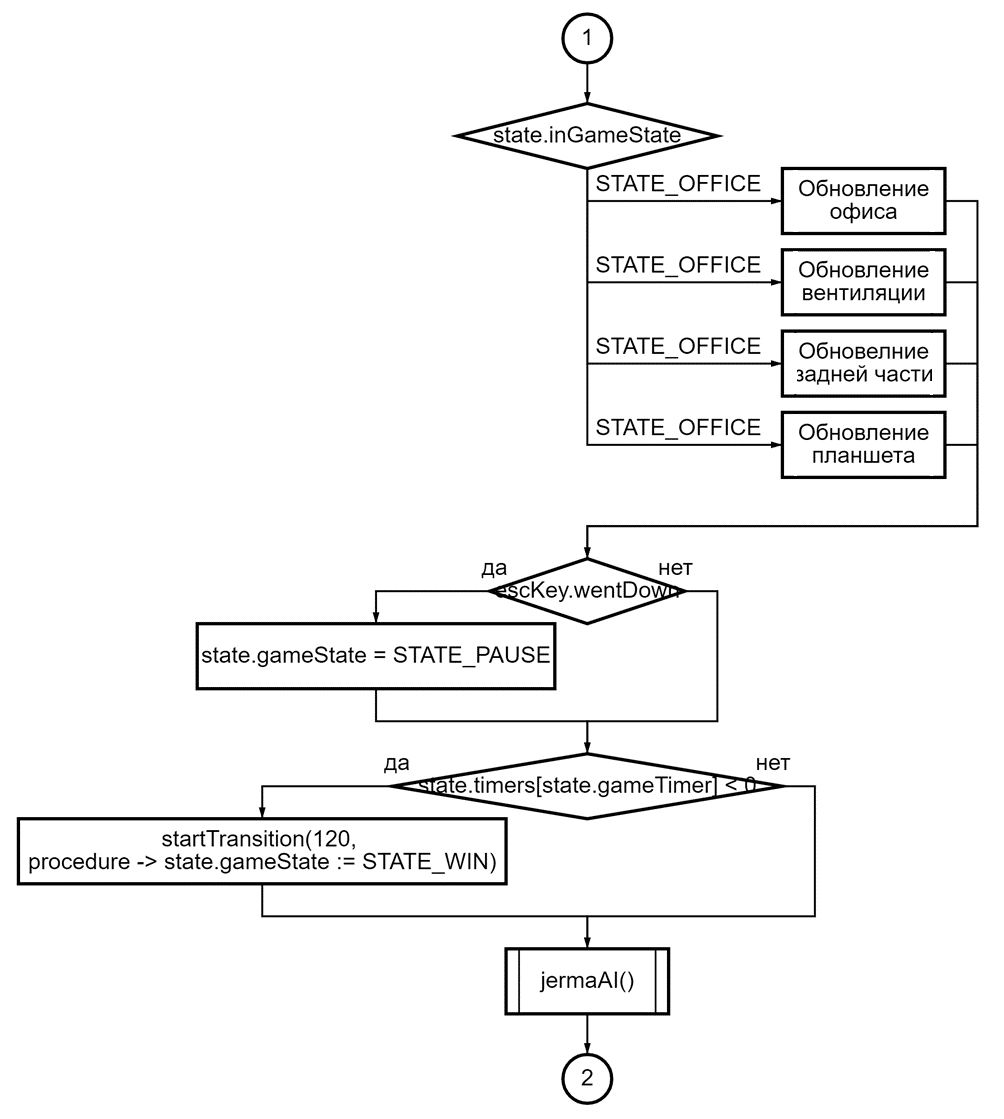
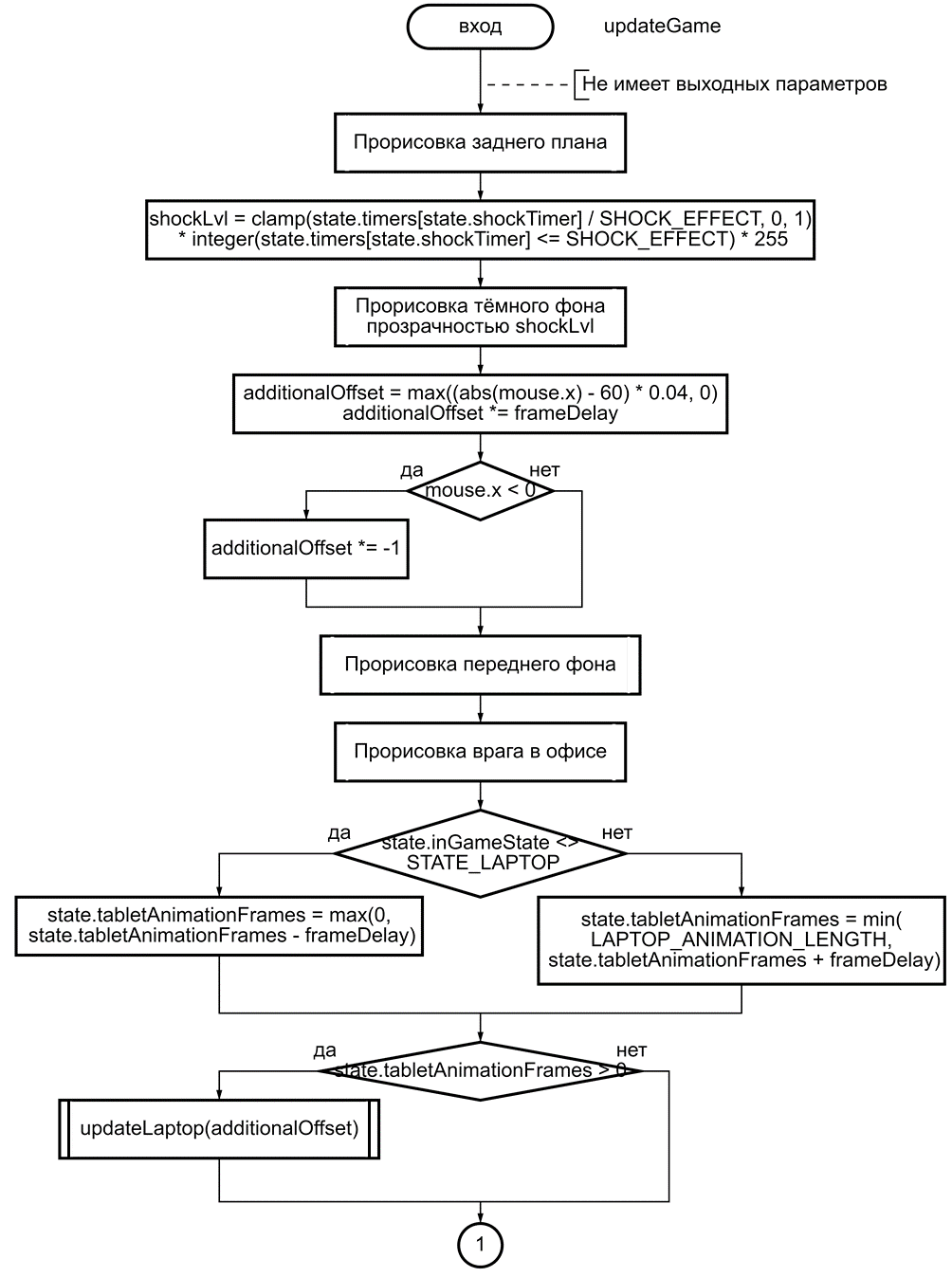
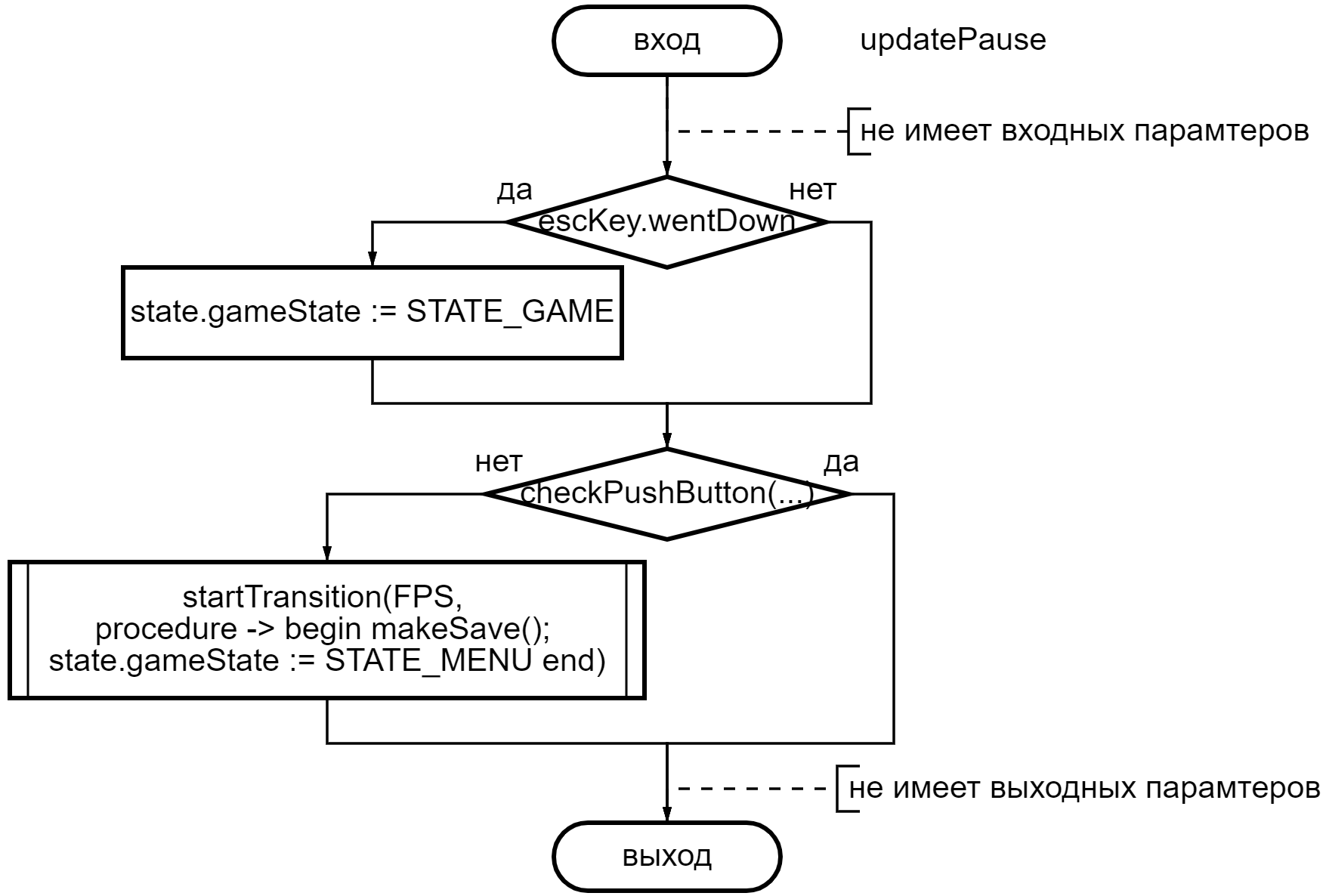
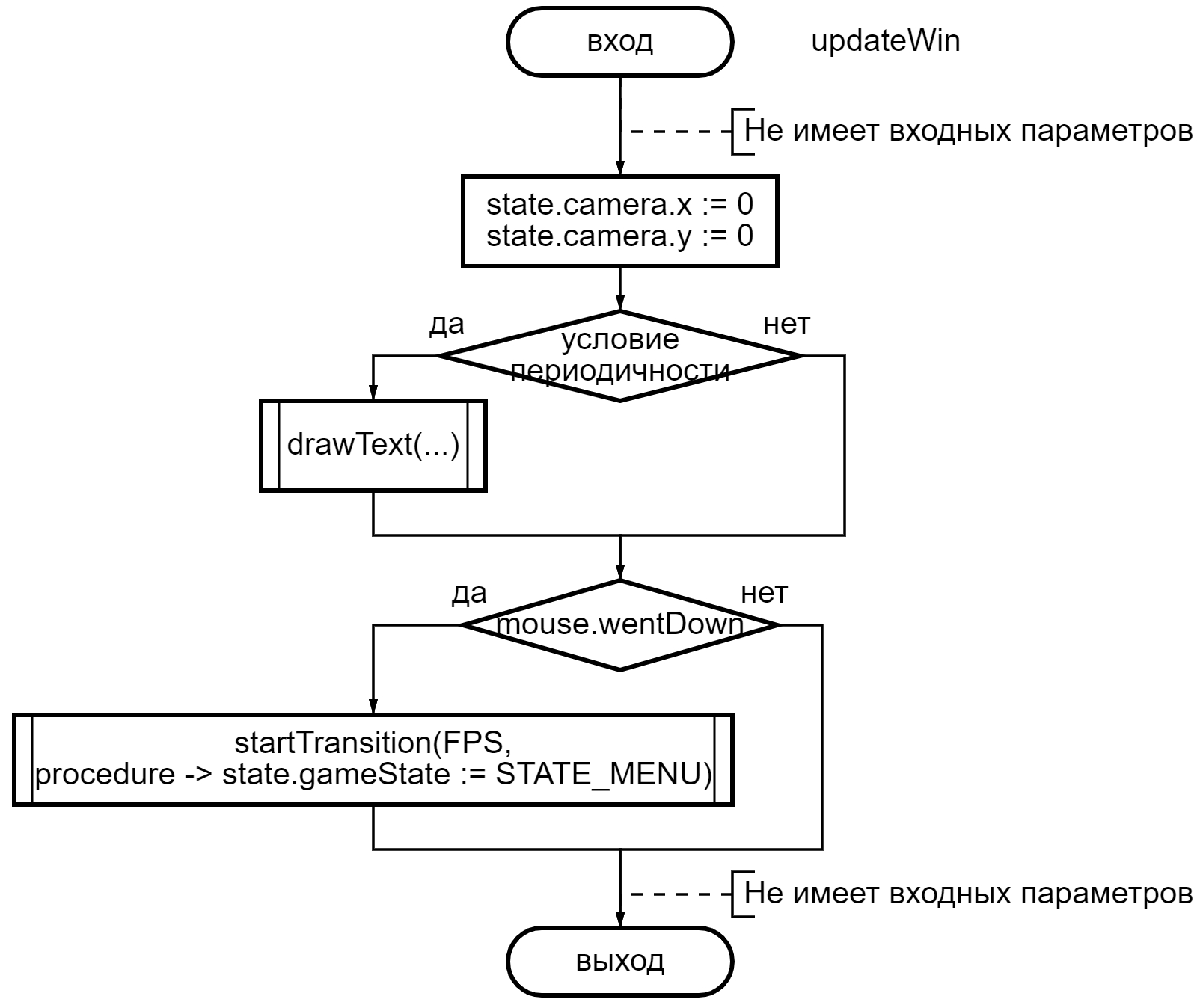
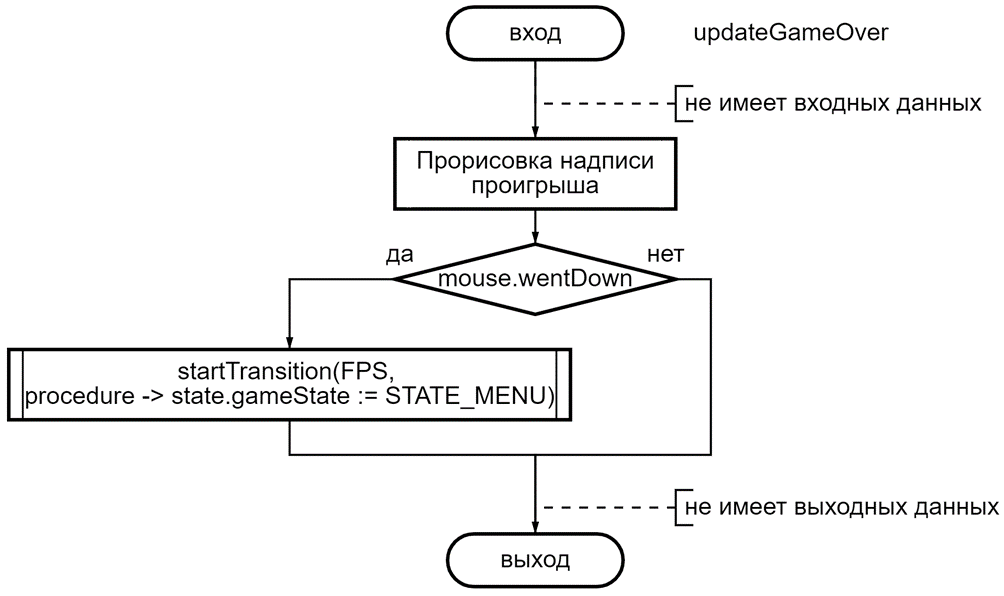
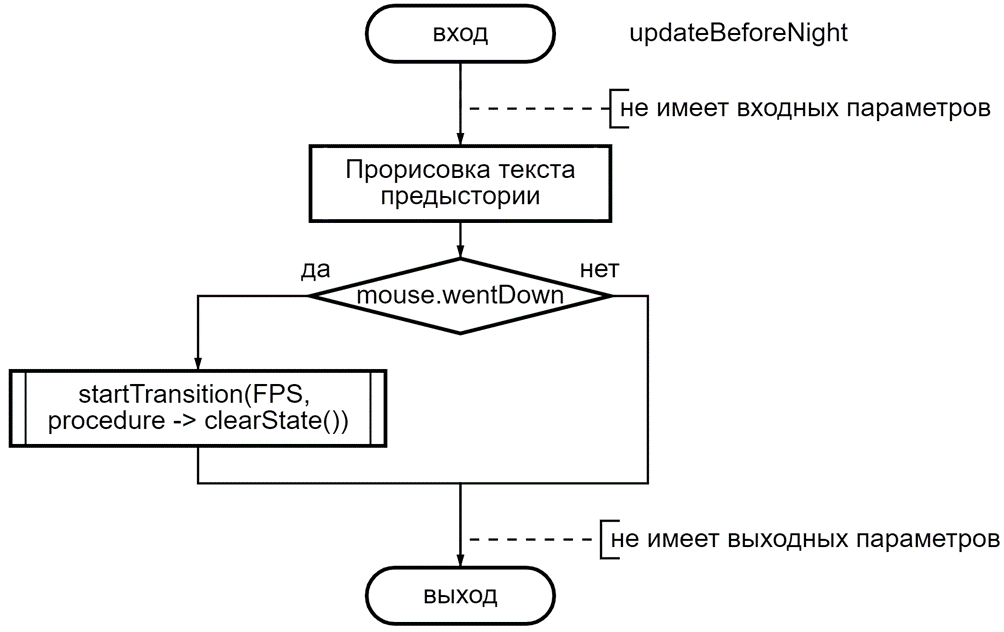
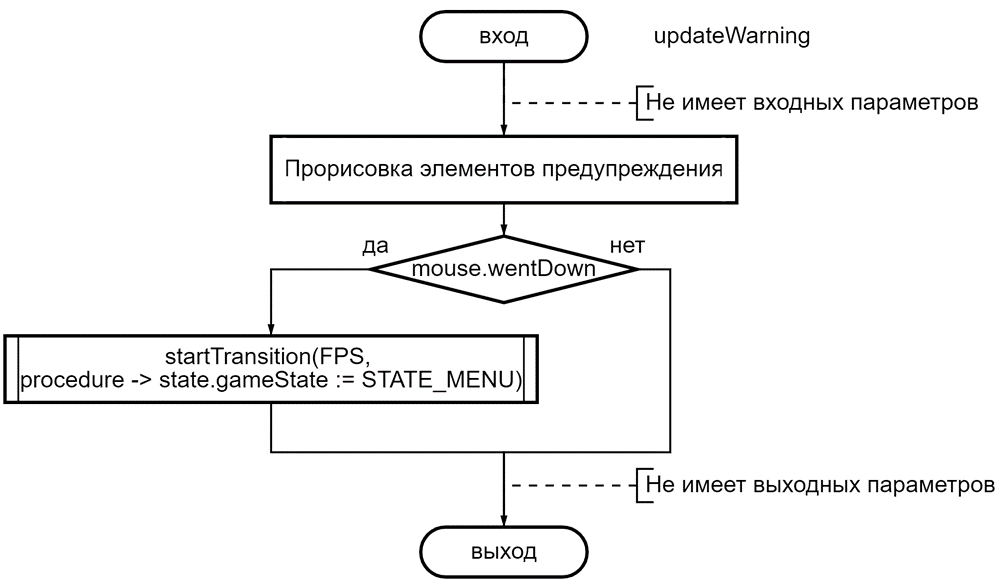
На экран выводится надпись «Вы умерли»

1. Состояние победа

На экран выводится мигающая надпись «6 AM» (6 часов утра)

1. Разработка схемы алгоритма и её описание



5555555555555555555555

1. Разработка схемы алгоритма и её описание

Этапы разработки игры:

1. Создание цикличности
2. Добавление ввода с мышки и клавиатуры
3. Создание функций прорисовки относительно размеров экрана и записи «камера»
4. Добавление игровых состояний и соответствующий подпрограмм их обновления
5. Добавление искусственного интеллекта
6. Создание игрового меню
7. Добавление визуальных эффектов

Основные подпрограммы:

**Модуль «Таймеры»** позволяет добавлять таймеры в массив, уменьшать их значения и измерять время

getTime – функция, возвращающая время компьютера в миллисекундах

addTimer – добавление нового таймера в массив таймеров

updateTimers – процедура обновления (уменьшения) таймеров

**Модуль «Математика»** содержит функции для базовых математических вычислений

getRandomFloat, getRandomInt – функции получения случайных чисел в диапазоне  
clamp – функция, помещающая число в диапазон

distanceBetweenPoints – функция получения расстояния между точками с заданными координатами

**Модуль «Прорисовка»** содержит функции прорисовки, действующие относительно эталонного разрешения экрана и камеры

drawRect, drawCircle, drawSprite, drawText, drawParagraph – процедуры прорисовки, учитывающие размеры экрана и рисующие относительно координат объекта «камера»

**Модуль «Ввод»** обрабатывает нажатия на мышь и клавиши клавиатуры

MouseDown, MouseMove, MouseUp, KeyDown, KeyUp – процедуры событий мышки и клавиатуры, меняющие глобальные переменные, отвечающие за ввод

clearInput – процедура очистки переменных ввода

**Модуль «Кнопки»** – функции обновления кнопок

checkActiveZone – функция, возвращающая true при введении мышки в область

checkButtonZone – функция проверки попадания координаты в зону

checkNormalButton – обновления стандартной кнопки меню

checkPushButton – обновления кнопки с зажатием

**Модуль «Визуал»** – процедуры добавления и обработки эффектов помех, шумов и перехода через затемнение

nothing – пустая процедура для создания лямбда-выражений, передающихся в startTransition

drawStatic – прорисовка визуального эффекта помех

updateNoise – прорисовка визуального эффекта шумов

makeNoise – добавление эффекта шума

startTransition – процедура начала перехода с затемнением

updateTransition – процедура прорисовки перехода с затемнением

**Модуль «Искусственный интеллект»** – для обработки поведения врага

initiateJumpscare – функция начала скримера

jermaNextTimer – функция возвращающая случайное число для таймера ходов ИИ

jermaDirection – функция перемещения ИИ

jermaAI – процедура обновления ИИ врага

**Модуль «Сохранения»** – для загрузки и сохранения незавершённой игры

loadSave – процедура загрузки сохранения из файла

makeSave – функция записи сохранения состояния игры в файл

**Модуль «Игра»** содержит процедуры обновления кадра игры и процедуры обновления и изменения состояний игры

cycle – процедура, передающаяся в beginFrameBasedAnimation, то есть повторяющаяся каждый кадр

changeState – процедура изменения состояния игры с соответствующим перемещением «камеры»

updateLaptop – функция прорисовки и обновления планшета и кнопок, доступных в нём

1. Инструкция по использованию разработанной программы

Программа работает на операционной системе «Windows».

При запуске программы пользователь видит экран предупреждения. После нажатия кнопки мыши игра переходит на экран меню. На экране меню доступно 3 кнопки (рисунок 1). Чтобы задействовать эффект кнопок необходимо навестись на надпись мышкой и удерживать её, пока не начнётся переход в следующее состояние. При нажатии на кнопку «Start game» игра переходит на экран оповещения, описывающий предысторию игры. При нажатии на мышку программа переходит в состояние игры.



Рисунок 1 – главное меню

В состоянии игры игрок может контактировать с активными зонами по бокам экрана, наводя на них мышкой, чтобы переходить по игровым состояниям и получать доступ к разным действиям. Первоначальное состояние называется «офис». В этом состоянии игрок может контактировать с двумя активными зонами (сверху и снизу экрана). Зона сверху экрана переводит игру в состояние «вентиляция». Зона снизу экрана переводит игру в состояние «планшет». Также в состоянии «офис» игрок может контактировать с дверями по бокам экрана. Нажатия на них мышкой отгоняют врагов, находящихся за дверьми. Между нажатиями на дверь существует перезарядка, вовремя которой вы не можете использовать нажатие на дверь или вентиляцию. Заряд можно увидеть на шкале-индикаторе в левой верхней части экрана.

В состоянии «вентиляция» пользователь может контактировать с активной зоной в нижней части экрана, чтобы вернуться в состояние «офис». Так же он может контактировать с изображением вентиляции, которая работает аналогично дверям в состоянии «офис».

В состоянии «планшет» пользователь может контактировать с активной зоной в нижней части экрана, чтобы вернуться в состояние «офис». Кроме того, игрок может нажимать на объекты, отмеченные на изображении карты здания в правой нижней части экрана, чтобы изменять отображаемое изображение камеры на экране. Зажимая любое другое место экрана, игрок может поворачивать камеру вправо и влево, чтобы просматривать её полностью.

Справа сверху отображено игровое время. Когда время достигнет значения «6:00» вы выигрываете.

На одной из камер находится изображение врага. Враг может перемещаться только по комнатам, соединённым на карте. Если враг находится на смежной камере в комнате, отмеченной, как камера, в которой находится персонаж игры, то он может при следующем передвижении оказаться в офисе. Он может появиться в одном из проходов (за одной из дверей или вентиляцией), ведущей в комнату, из которой враг зашёл в офис (если со 2-ой, то в левой двери, если с 1-ой, то в правой двери, если с 9-ой или 7-ой, то с вентиляции). Карту камер можно видеть на рисунке 2. Поэтому необходимо следить за его перемещениями, чтобы знать, с какой стороны он заходит. Чтобы прогнать его на какое-то время нужно нажать на соответствующую дверь или вентиляцию, за которой он стоит.

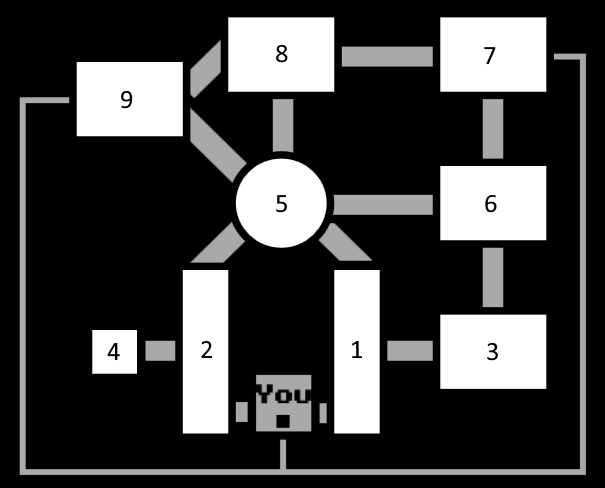


Рисунок 2 – Карта здания с пронумерованными камерами

Во время экрана игры пользователь может нажать на клавишу Escape, чтобы перейти на экран «пауза». На данном экране пользователь может контактировать с кнопкой «Save & exit» (сохраниться и выйти), чтобы сохранить свой прогресс в текущей игре и выйти в меню. Также игрок может повторно нажать клавишу Escape, чтобы вернуться на экран игры. Кроме того, сохранение прогресса происходит автоматически при выходе из игры.

При удержании кнопки «Continue» (продолжить) экрана меню, программа перейдёт на экран игры с того момента, когда было сделано последнее сохранение.

При удержании кнопки «Exit» (выход) программа будет закрыта.

1. Результаты отладки
2. После доработки функции jermaDirection, определяющей следующее местоположение врага было замечено, что при попытках перемещения врага программа вылетает, и враг иногда продвигается между несмежными камерами, возвращая в консоль предупреждение о том, что индекс находился вне границ массива (рисунок 3).

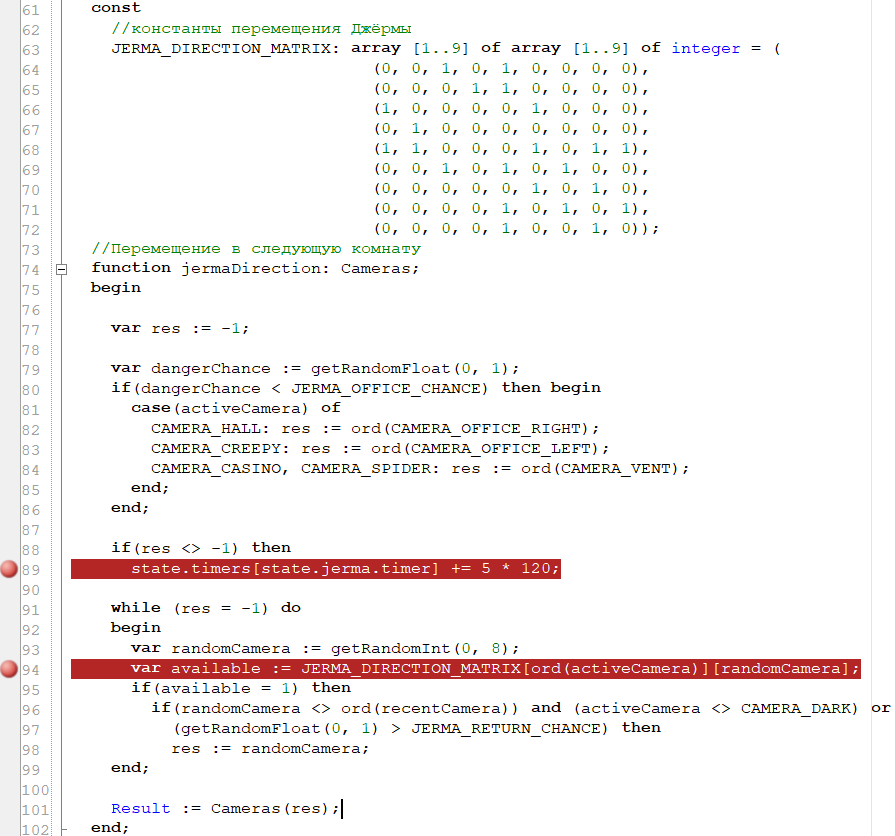


Рисунок 3 – место отладки 1

Для отладки было решено поставить точки отладки на местах, где берутся данные из массивов. Во время отладки оказалось, что ошибка появляется в строке 94 при значении переменной randomCamera равном 0.

Было замечено, что нумерация массива JERMA\_DIRECTION\_MATRIX начиналась не с нуля. После исправления пределов изменения индексов ошибка больше не наблюдалась.

1. При создании не работает функция checkActiveZone, проверяющая введение мышки в область.

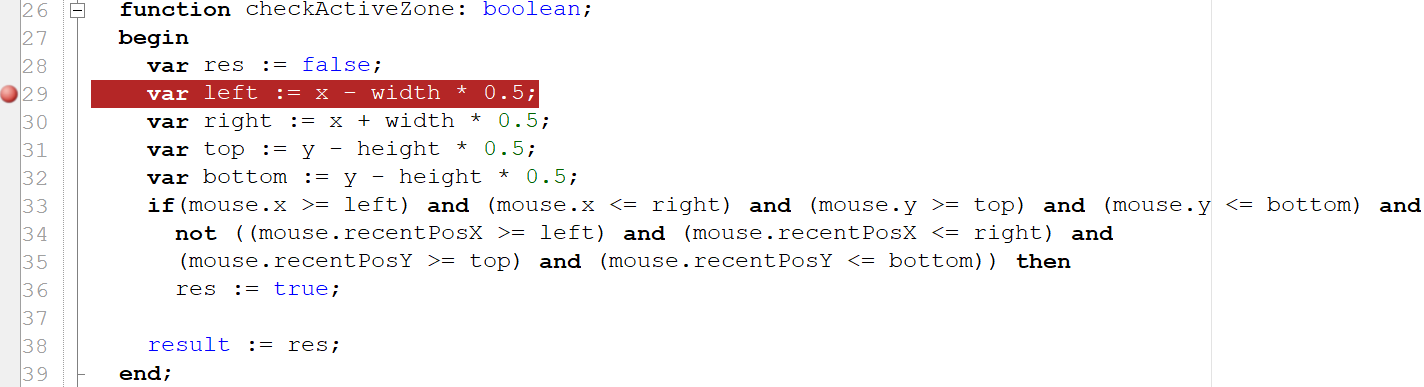


Рисунок 4 – место отладки 2

При отладке было выявлено, что значения переменных top и bottom совпадают. При повторном рассмотрении оказалось, что значение bottom было задано неправильно. Оно было поправлено на y + height \* 0.5. После исправления проблем не наблюдалось.

1. После переработки функции makeSave сохранения игры перестали работать.

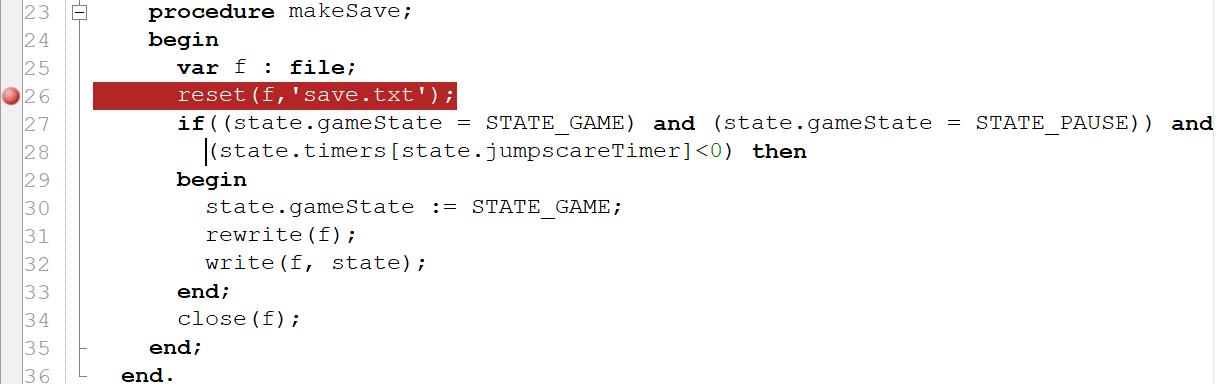


Рисунок 5 – место отладки 3

Во время отладки было выявлено, что программа не проходит условие на строке 27 (рисунок 5). При рассмотрении было выявлено, что условие в любом случае возвращает false. Условие было изменено на (state.gameState = STATE\_GAME) and (state.gameState = STATE\_PAUSE)) and (state.timers[state.jumpscareTimer]<0.

1. Решение контрольных примеров

Тестирование функции «Взаимодействие с элементами меню»

Ожидаемая реакция: при наведении мышки на кнопку меню и продолжительном нажатии на кнопку мыши кнопка меню начинает трястись, показывая прогресс зажатия. После этого должен начаться переход с затемнением с последующим выполнением эффекта кнопки.

Фактическая реакция: изображение кнопки начинает трястись. После секундного удерживания кнопки экран начинает затемняться. После этого игра переходит в следующее состояние (или закрывается, если это кнопка выхода). Начало эффекта затемнения можно наблюдать на рисунке 6.



Рисунок 6 – начало перехода по кнопке меню

Вывод: ожидаемая реакция и фактическая реакция совпадают, следовательно, функция работает корректно.

Тестирование функции «Cохранение»

Ожидаемая реакция: после выхода из программы при нажатии на кнопку продолжения на экране меню игра начнётся с того момента, где был произведён выход.

Фактическая реакция: во время выхода из программы игра находится в состоянии «планшет» и активирована камера 6 (рисунок 7).

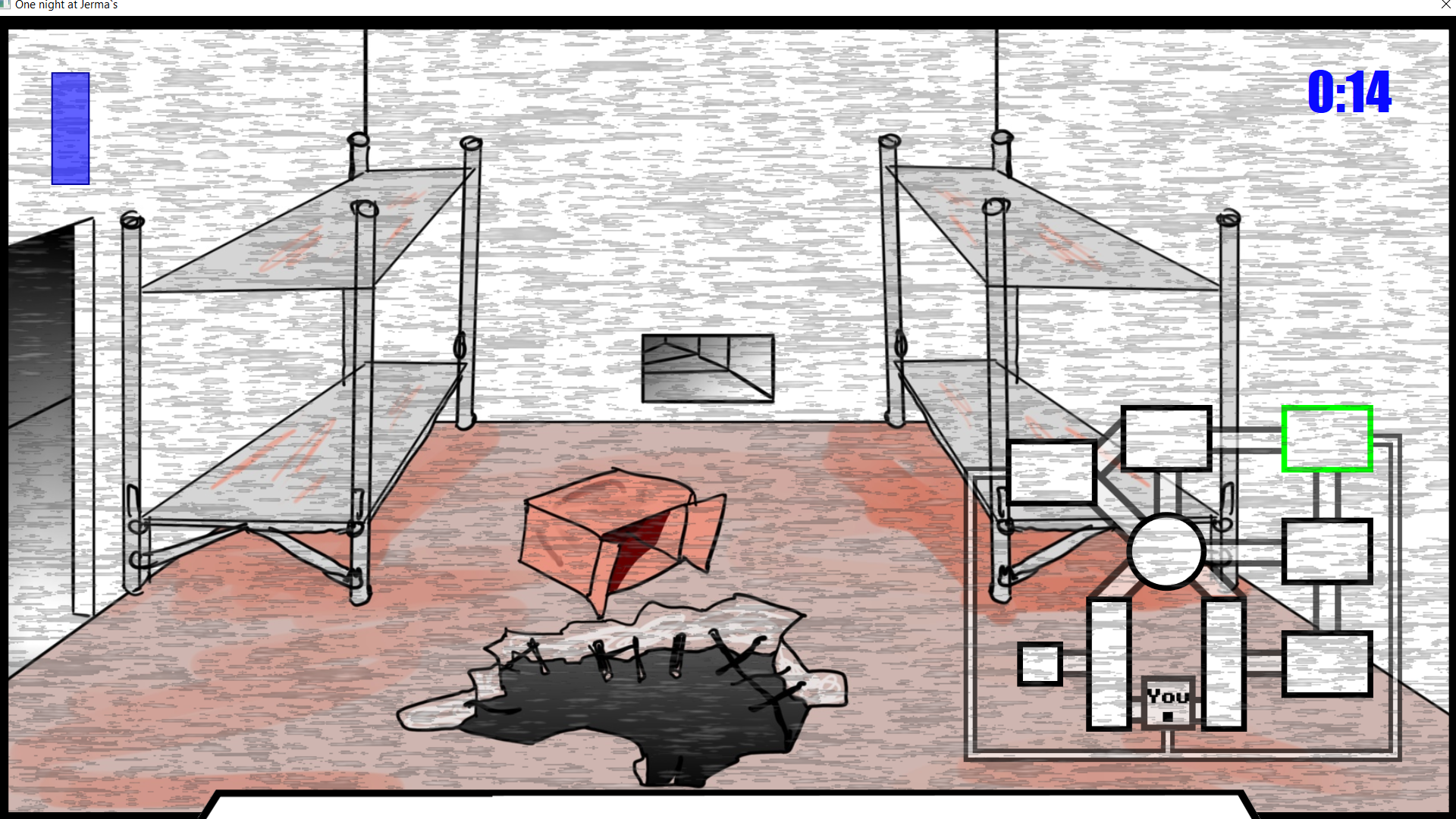


Рисунок 7 – состояние игры до выхода

После этого заново включая игру и активируя кнопку продолжения игры, игра начинается с того момента, на котором мы вышли из игры (рисунок 8).

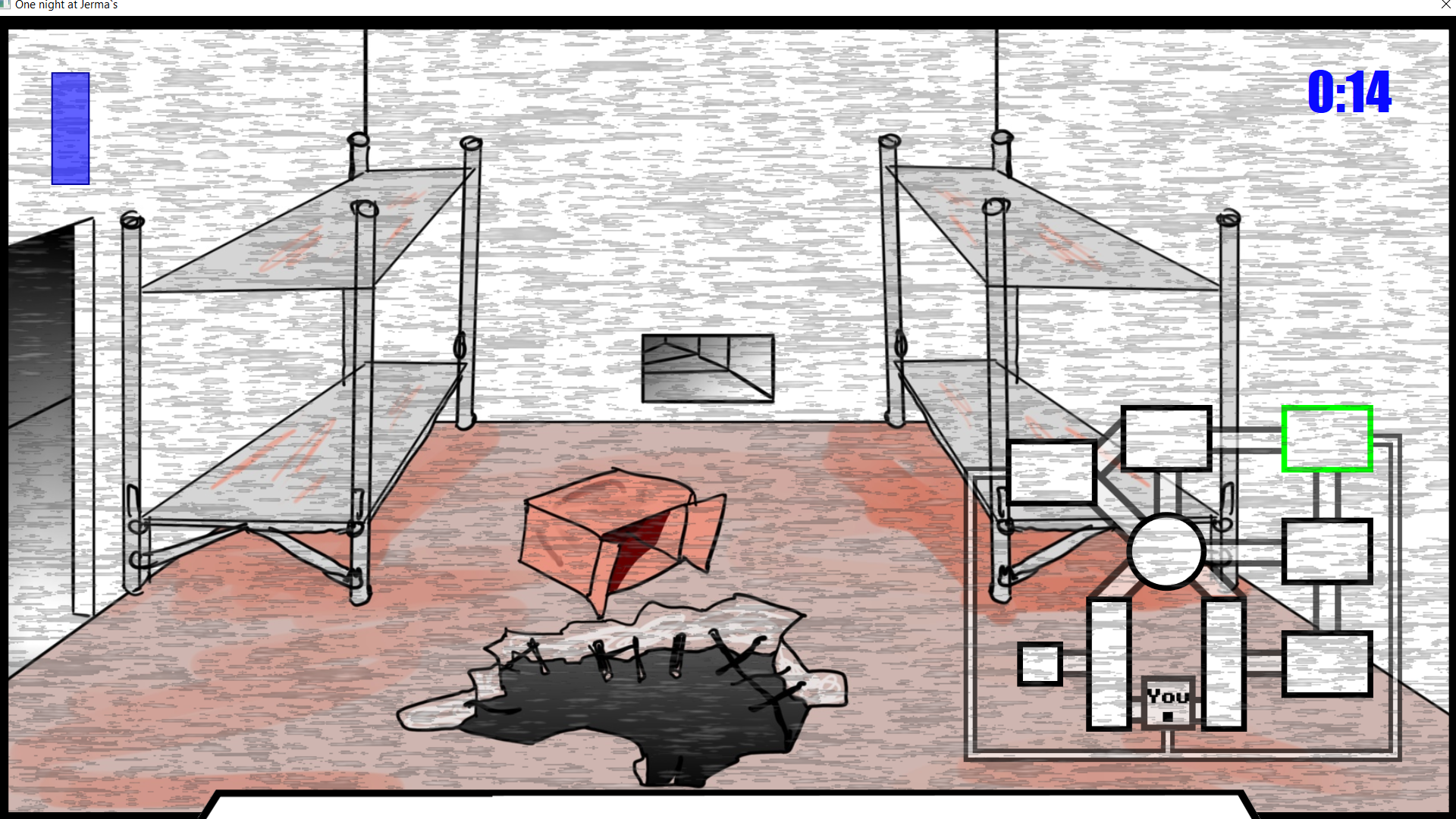


Рисунок 8 – состояние игры после загрузки сохранения

Состояния совпадают, следовательно, сохранения игры прошло успешно.

Вывод: ожидаемая и фактическая реакции совпадают

Тестирование функции «Пауза»

Ожидаемая реакция: при нажатии на экране игры кнопки клавиатуры Escape открывается экран паузы. При повторном нажатии снова выводится экран игры. При этом состояние игры до нажатия на паузу идентично состоянию после выхода с экрана паузы.

Фактическая реакция: изображение до нажатия на кнопку паузы можно видеть на рисунок 9.

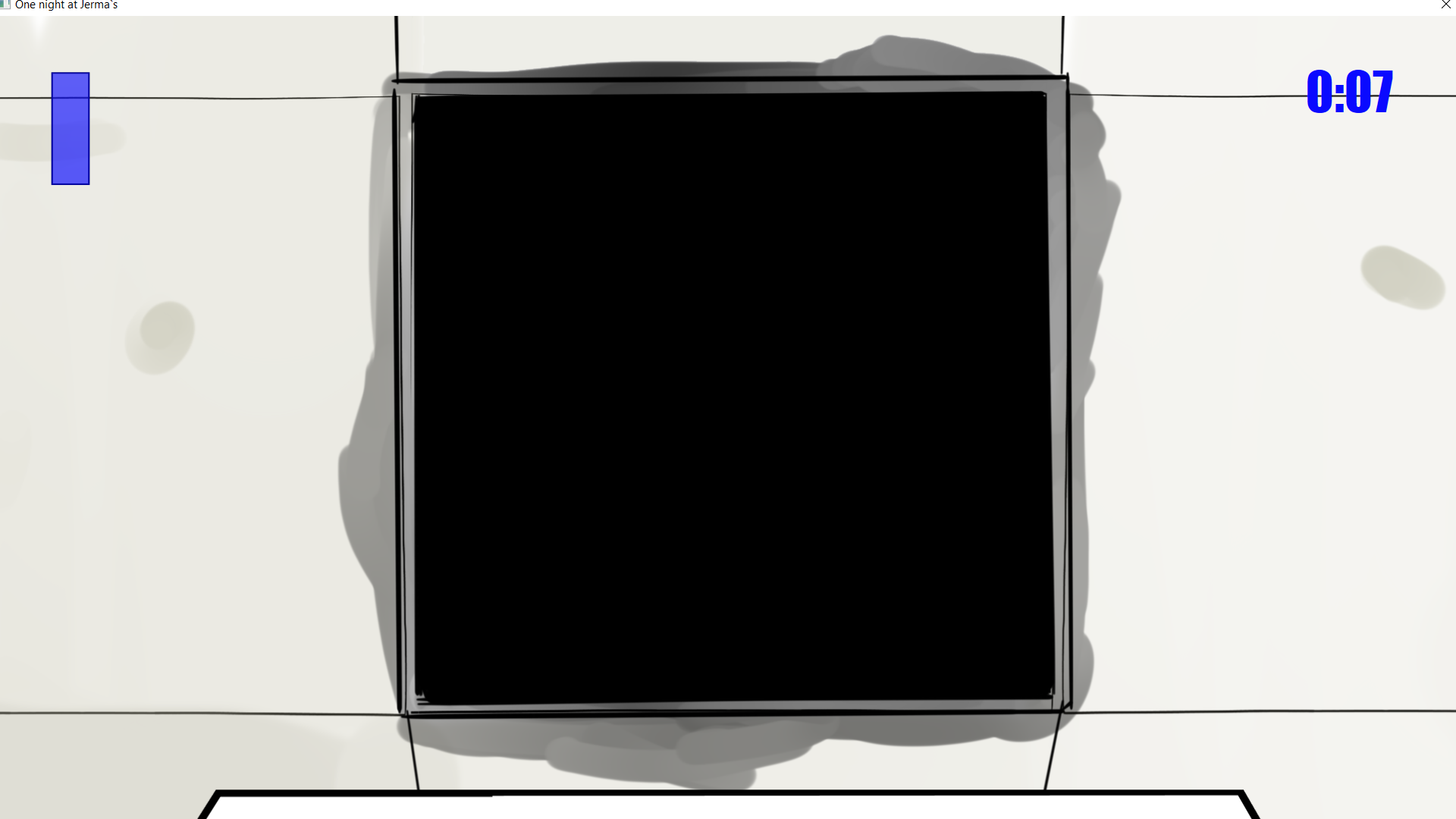


Рисунок 9 – изображение до паузы

После выхода из состояния «пауза» изображение не изменилось.

Вывод: Ожидаемая и фактическая реакции совпадают.

1. Календарь выполнения работ

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Этапы решения задачи | Ориентировочная трудоёмкость, ч | Реально затрачиваемое время, ч |
| 1 | Анализ предметной области и формализация задачи | 8 | 4 |
| 2 | Разработка структуры организации данных | 8 | 4 |
| 3 | Составление алгоритма | 12 | 8 |
| 4 | Написание программы | 8 | 80 |
| 5 | Подготовка тестов | 8 | 2 |
| 6 | Устранение синтаксических ошибок | 4 | 4 |
| 7 | Тестирование и отладка | 24 | 30 |
| 8 | Подготовка документации | 8 | 18 |
| Итого: | | 80 | 150 |

Заключение

В результате выполнения данной работы была создана игра с функциями меню, паузы, системы сохранения прогресса, масштабирования изображений и контроля скорости действий в зависимости от количества кадров в секунду. Была создана удобная среда для прорисовки, обработки ввода с клавиатуры и мышки, система состояний, позволяющая удобно добавлять новые экраны и переходы между ними. С помощью них удалось создать удачную визуальную часть игры, а также функции сохранения прогресса игры, паузы и меню.

Все поставленные цели были выполнены. Был создан полноценный игровой движок, а на его основе уже полноценная игра.

Список используемой литературы

1. Конструирование программ: методические указания к курсовой работе**/** Рязан. радиотехн. ин-т; сост.: В.С. Новичков, О.Г. Швечкова. – Рязань, 1992. №2133.
2. Структурированные типы данных в Турбо Паскале: метод. указ. к лаб. раб.**/** Под ред. В.С. Новичкова. – Рязань, 2001. №3193.
3. Алгоритмические языки и программирование: методические указания к курсовой работе**/** Рязан. гос. Р
4. Справка PascalABC.net

Приложение

{$F+}

**program** main;

//бибилиотеки прорисовки и звука

**uses** GraphWpf;

**uses** game,saveState,visuals,timer,input,saves;

**begin**

//подготовка к старту игры

//эффект затемнения, который начинается с темноты

startTransition(120, **procedure**->nothing());

state.timers[transition.timer] := 0;

//задаются начальные значения для переменных, необходимых до начала основной игры

state.gameState := STATE\_WARNING;

//параметры окна

window.Caption := 'One night at Jerma`s';

window.SetSize(System.Windows.Forms.Screen.PrimaryScreen.Bounds.Width,

System.Windows.Forms.Screen.PrimaryScreen.Bounds.Height);

window.IsFixedSize := true;

//задаётся большое значение первого кадра, чтобы при вычислении frameDelay результат был равен 0

recentFrame := getTime() + 100000;

//задаём процедуры событий

OnMouseDown := mouseDown;

OnMouseMove := mouseMove;

OnMouseUp := mouseUp;

OnKeyDown := keyDown;

OnKeyUp := keyUp;

OnClose := makeSave;

BeginFrameBasedAnimation(cycle, 120);

**end**.

//Модуль "Математика"

**unit** math;

**interface**

//функции выбора случайного числа из диапазона

**function** getRandomFloat(min, max: real): real;

**function** getRandomInt(min, max: integer): integer;

//clamp помещает число в промежуток

**function** clamp(val, minv, maxv: real): real;

//расстояние между точками с заданными координатами

**function** distanceBetweenPoints(x1, y1, x2, y2:real): real;

**implementation**

**function** getRandomFloat: real;

**begin**

Result := random() \* (max - min) + min;

**end**;

**function** getRandomInt: integer;

**begin**

Result := round(getRandomFloat(min - 0.5, max + 0.49));

**end**;

**function** clamp: real;

**begin**

Result := min(maxv, max(val, minv));

**end**;

**function** distanceBetweenPoints: real;

**begin**

Result := sqrt(sqr(x1 - x2) + sqr(y1 - y2));

**end**;

**begin**

randomize;

**end**.

//Модуль "Прорисовка" содержит функции прорисовки, действующие относительно

//эталонного разрешения экрана и камеры, положение которой можно изменять

**unit** drawing;

**interface**

**uses** graphWpf;

//камера, относительно которой происходит прорисовка

**type**

CameraType = **record**

x: real;

y: real;

targetX: real;

targetY: real;

**end**;

Color = GColor;

Alignment = Alignment;

**const**

//эталонные величины

W\_WIDTH = 1920;

W\_HEIGHT = 1080;

FPS = 120;

//скорость перемещения камеры

CAMERA\_SPEED = 0.08;

//стандартный шрифт

DEAFAULT\_FONT = 'True Lies';

//функции прорисовки с учётом размеров окна

**procedure** drawSprite(camera: CameraType; src: string; x, y, width, height: real);

**procedure** drawRect(camera: CameraType; x, y, width, height: real; clr: Color := RGB(255, 255, 255));

**procedure** drawCircle(camera: CameraType; x, y, radius: real; clr: Color := RGB(255, 255, 255));

**procedure** drawText(camera: CameraType; x, y, width, height: real; text: string; color: GColor := RGB(255, 255, 255);

fontStr: string := DEAFAULT\_FONT; align: Alignment := center);

**procedure** drawParagraph(camera: CameraType; x, y, width, height: real; indent: real; text: string;

color: GColor := RGB(255, 255, 255); fontStr: string := DEAFAULT\_FONT; align: Alignment := center);

**var** ARGB: **function**(a,r,g,b:byte):Color = ARGB;

**var** RGB: **function**(r,g,b:byte):Color = RGB;

**implementation**

**procedure** drawSprite;

**begin**

src := './bitmaps/' + src;

x -= camera.x - W\_WIDTH \* 0.5;

y -= camera.y - W\_HEIGHT \* 0.5;

x := window.Width \* 0.5 + (x - W\_WIDTH \* 0.5) \* window.Width / W\_WIDTH;

y := window.Height \* 0.5 + (y - W\_HEIGHT \* 0.5) \* window.Height / W\_HEIGHT;

width \*= window.Width / W\_WIDTH;

height \*= window.Height / W\_HEIGHT;

drawImage(x - width \* 0.5, y - height \* 0.5, width, height, src);

**end**;

**procedure** drawRect;

**begin**

x -= camera.x - W\_WIDTH \* 0.5;

y -= camera.y - W\_HEIGHT \* 0.5;

x := window.Width \* 0.5 + (x - W\_WIDTH \* 0.5) \* window.Width / W\_WIDTH;

y := window.Height \* 0.5 + (y - W\_HEIGHT \* 0.5) \* window.Height / W\_HEIGHT;

width \*= window.Width / W\_WIDTH;

height \*= window.Height / W\_HEIGHT;

fillRectangle(x - width \* 0.5, y - height \* 0.5, width, height, clr);

**end**;

**procedure** drawCircle;

**begin**

x -= camera.x - W\_WIDTH \* 0.5;

y -= camera.y - W\_HEIGHT \* 0.5;

x := window.Width \* 0.5 + (x - W\_WIDTH \* 0.5) \* window.Width / W\_WIDTH;

y := window.Height \* 0.5 + (y - W\_HEIGHT \* 0.5) \* window.Height / W\_HEIGHT;

**var** radiusX := radius \* window.Width / W\_WIDTH;

**var** radiusY := radius \* window.Height / W\_HEIGHT;

fillEllipse(x, y, radiusX, radiusY, clr);

**end**;

**procedure** drawText;

**begin**

x -= camera.x - W\_WIDTH \* 0.5;

y -= camera.y - W\_HEIGHT \* 0.5;

x := window.Width \* 0.5 + (x - W\_WIDTH \* 0.5) \* window.Width / W\_WIDTH;

y := window.Height \* 0.5 + (y - W\_HEIGHT \* 0.5) \* window.Height / W\_HEIGHT;

width \*= window.Width / W\_WIDTH;

height \*= window.Height / W\_HEIGHT;

Font.Color := color;

Font.Size := height;

Font.Name := fontStr;

drawText(x - width \* 0.5, y - height \* 0.5, width, height, text, align);

**end**;

**procedure** drawParagraph;

**begin**

x -= camera.x - W\_WIDTH \* 0.5;

y -= camera.y - W\_HEIGHT \* 0.5;

x := window.Width \* 0.5 + (x - W\_WIDTH \* 0.5) \* window.Width / W\_WIDTH;

y := window.Height \* 0.5 + (y - W\_HEIGHT \* 0.5) \* window.Height / W\_HEIGHT;

width \*= window.Width / W\_WIDTH;

height \*= window.Height / W\_HEIGHT;

indent \*= window.Height / W\_HEIGHT;

indent \*= window.Width / W\_WIDTH;

**var** words := text.Split((' '));

**var** str := '';

Font.Color := color;

Font.Size := height;

Font.Name := fontStr;

**for var** wordIndex := 0 **to** length(words) - 1 **do**

**begin**

**var** textWidth := TextWidth(str + words[wordIndex]);

**if**(textWidth <= width) **then**

str += words[wordIndex] + ' '

**else begin**

drawText(x - width \* 0.5, y - height \* 0.5, width, height, str, align);

y += height \* indent;

str := words[wordIndex] + ' ';

**end**;

**end**;

drawText(x - width \* 0.5, y - height \* 0.5, width, height, str, align);

**end**;

**end**.

//Модуль "Ввод" добавляет процедуры обработки нажатий на мышь и клавиши клавиатуры

**unit** input;

**interface**

**uses** graphWpf;

**const**

//эталонные величины

W\_WIDTH = 1920;

W\_HEIGHT = 1080;

**type**

//тип "мышь"

MouseType = **record**

x: real;

y: real;

gamePosX: real;

gamePosY:real;

recentPosX: real;

recentPosY:real;

//нажата

isDown: boolean;

//нажалась

wentDown: boolean;

//отжалась

wentUp: boolean;

**end**;

KeyboardKey = **record**

wentDown: boolean;

isDown: boolean;

wentUp: boolean;

**end**;

//мышка

**var** mouse: MouseType;

escKey: KeyboardKey;

//процедуры событий мышки и нажатий

**procedure** MouseDown(x, y: real; mb: integer);

**procedure** MouseMove(x, y: real; mb: integer);

**procedure** MouseUp(x, y: real; mb: integer);

**procedure** ClearInput();

**procedure** KeyDown(k: Key);

**procedure** KeyUp(k: Key);

**implementation**

//процедуры событий мышки и нажатий

**procedure** MouseDown;

**begin**

mouse.isDown := true;

mouse.wentDown := true;

**end**;

**procedure** MouseMove;

**begin**

mouse.x := x;

mouse.y := y;

mouse.x -= window.Width \* 0.5;

mouse.y -= window.Height \* 0.5;

mouse.x \*= W\_WIDTH / window.Width;

mouse.y \*= W\_HEIGHT / window.Height;

**end**;

**procedure** MouseUp;

**begin**

mouse.isDown := false;

mouse.wentUp := true;

**end**;

**procedure** ClearInput;

**begin**

mouse.wentDown:=false;

mouse.wentUp:=false;

escKey.wentDown:=false;

escKey.wentUp:=false;

**end**;

**const**

VK\_ESC = 13;

**procedure** KeyDown;

**begin**

**if**(integer(k) = VK\_ESC) **then begin**

escKey.wentDown:=true;

escKey.isDown:=true;

**end**;

**end**;

**procedure** KeyUp;

**begin**

**if**(integer(k) = VK\_ESC) **then begin**

escKey.wentUp:=true;

escKey.isDown:=false;

**end**;

**end**;

**end**.

//Модуль "Сохранённое состояние" содержит параметры игры, которые

//сохраняются в файле

**unit** saveState;

**interface**

**uses** drawing, input, timer;

**type**

//типы игровых состояний

GameStateType = (STATE\_WARNING, STATE\_MENU, STATE\_BEFORE\_NIGHT, STATE\_GAME, STATE\_GAME\_OVER,STATE\_WIN, STATE\_PAUSE);

InGameStateType = (STATE\_OFFICE, STATE\_LAPTOP, STATE\_BACK, STATE\_VENT);

//тип отображения камеры на карте

MapCameraType = (CAMERA\_TYPE\_RECT, CAMERA\_TYPE\_ROUND);

//порядковый тип для каждой камеры на карте

Cameras = (CAMERA\_HALL, CAMERA\_CREEPY, CAMERA\_BATH, CAMERA\_DARK, CAMERA\_SOLAR, CAMERA\_CAGE, CAMERA\_SPIDER, CAMERA\_RING,

CAMERA\_CASINO, CAMERA\_OFFICE\_RIGHT, CAMERA\_OFFICE\_LEFT, CAMERA\_VENT);

JumpscareType = (JUMPSCARE\_NONE, JUMPSCARE\_JERMA, JUMPSCARE\_SPIDER, JUMPSCARE\_SUS, JUMPSCARE\_RAT);

//тип отображаемой камеры

MapCameraObjType = **record**

x:real;

y:real;

width:real;

height: real;

cType: MapCameraType;

color: Color;

**end**;

//сохраняемая запись состояния игры

StateType = **record**

//таймеры и размер массива таймеров

timersLength: integer;

timers: TimersType;

//состояние программы

gameState: GameStateType;

inGameState: InGameStateType;

//смещения рисунков для эффекта поворота вправо и влево

cameraOffset: real;

//задействованая камера

currentCamera: integer;

//камера

camera: CameraType;

//уровень поднятия планшета

tabletAnimationFrames: real;

//объект монстра Джёрма

jerma: **record**

recentCamera: Cameras;

camera: Cameras;

timer: integer;

lvl: integer;

**end**;

//переменные для скримера

jumpscare: jumpscareType;

jumpscareTimer: integer;

//переменные для отпугивания

shockTimer: integer;

shock: Cameras;

shockReloadTimer: integer;

//время

gameTimer: integer;

//шум

noise: **record**

posY: real;

sizeY: real;

timer: integer;

**end**;

**end**;

**var** state: StateType;

**implementation**

**begin**

state.noise.timer := addTimer(state.timers,state.timersLength,-1);

**end**.

{$F+}

//Модуль "Кнопки" содержит функции создания кнопок

**unit** buttons;

**interface**

**uses** drawing, math, input, saveState, visuals;

**var**

//время удерживания кнопки

btnPushing: real;

//проверка наведения на активную зону

//Активная зона - вид взаимодействия пользователя с программой, при котором пользователь наводит мышью на

//определённую область экрана, чтобы произошло событие.

**function** checkActiveZone(x,y,width,height: real; mouse: MouseType): boolean;

//проверка нажатия на кнопку

**function** checkButtonZone(x,y,width,height,mouseX,mouseY: real; condition: boolean := true): boolean;

//проверка нажатия на кнопку меню

**function** checkNormalButton(x,y, width,height,mouseX,mouseY: real; str: string; condition: boolean; align: Alignment): boolean;

**function** checkPushButton(x,y,width,height,mouseX,mouseY:real; str: string; condition: boolean; holdingLength: real; frameDelay: real; align: Alignment): boolean;

**implementation**

//проверка наведения на активную зону

//Активная зона - вид взаимодействия пользователя с программой, при котором пользователь наводит мышью на

//определённую область экрана, чтобы произошло событие.

**function** checkActiveZone: boolean;

**begin**

**var** res := false;

**var** left := x - width \* 0.5;

**var** right := x + width \* 0.5;

**var** top := y - height \* 0.5;

**var** bottom := y + height \* 0.5;

**if**(mouse.x >= left) **and** (mouse.x <= right) **and** (mouse.y >= top) **and** (mouse.y <= bottom) **and**

**not** ((mouse.recentPosX >= left) **and** (mouse.recentPosX <= right) **and**

(mouse.recentPosY >= top) **and** (mouse.recentPosY <= bottom)) **then**

res := true;

result := res;

**end**;

//проверка нажатия на кнопку

**function** checkButtonZone: boolean;

**begin**

**var** res := false;

**var** left := x - width \* 0.5;

**var** right := x + width \* 0.5;

**var** top := y - height \* 0.5;

**var** bottom := y + height \* 0.5;

**if**(mouseX >= left) **and** (mouseX <= right) **and** (mouseY >= top) **and** (mouseY <= bottom) **and** condition **then**

res := true;

result := res;

**end**;

//проверка нажатия на кнопку меню

**function** checkNormalButton: boolean;

**begin**

**var** res := false;

**if**(checkButtonZone(x,y, width,height, mouseX,mouseY, state.timers[transition.timer]<=0)) **then**

**begin**

**if**(getRandomFloat(0, 1) > 0.95) **then**

str[getRandomInt(1, length(str))] := chr(getRandomInt(0, 90));

**if**(getRandomFloat(0, 1) > 0.95) **then begin**

x += getRandomFloat(-5, 5);

y += getRandomFloat(-5, 5);

**end**;

**if**(condition) **then**

res := true;

**end**;

drawText(state.camera, x,y, width,height, str, ARGB(255,255, 255, 255), DEAFAULT\_FONT, align);

result := res;

**end**;

**function** checkPushButton: boolean;

**begin**

**var** res := false;

**if**(checkButtonZone(x,y, width,height, mouseX,mouseY, state.timers[transition.timer]<=0)) **then**

**begin**

x+=getRandomInt(-10, 10)\* btnPushing / holdingLength;

y+=getRandomInt(-10, 10)\* btnPushing / holdingLength;

**if**(condition) **then begin**

btnPushing += frameDelay;

**if**(btnPushing >= holdingLength) **then**

res := true;

**end**;

**end**;

checkNormalButton(x,y, width,height, mouseX,mouseY, str, condition, align);

result := res;

**end**;

**begin**

btnPushing := 0;

**end**.

{$F+}

//Модуль "ИИ" для искусственного интеллекта

**unit** AI;

**interface**

**uses** saveState, timer, math,visuals;

**const**

JUMPSCARE\_LENGTH = 120;

//функции искусственного интеллекта монстров

//Монстр Джёрма

**procedure** jermaAI();

**implementation**

**const**

//константы перемещения Джёрмы

JERMA\_DIRECTION\_MATRIX: **array** [0..8] **of array** [0..8] **of** integer = (

(0, 0, 1, 0, 1, 0, 0, 0, 0),

(0, 0, 0, 1, 1, 0, 0, 0, 0),

(1, 0, 0, 0, 0, 1, 0, 0, 0),

(0, 1, 0, 0, 0, 0, 0, 0, 0),

(1, 1, 0, 0, 0, 1, 0, 1, 1),

(0, 0, 1, 0, 1, 0, 1, 0, 0),

(0, 0, 0, 0, 0, 1, 0, 1, 0),

(0, 0, 0, 0, 1, 0, 1, 0, 1),

(0, 0, 0, 0, 1, 0, 0, 1, 0));

JERMA\_MIN\_OPP\_TIME = 3\*120;

JERMA\_MAX\_OPP\_TIME = 6\*120;

JERMA\_RETURN\_CHANCE = 0.85;

JERMA\_OFFICE\_CHANCE = 0.75;

//проигрыш и скример

**procedure** initiateJumpscare(**var** jumpscare: JumpscareType; neededJumpscare: JumpscareType;

**var** jumpscareTimer: integer);

**begin**

**if**(jumpscare = JUMPSCARE\_NONE) **then begin**

jumpscare := neededJumpscare;

jumpscareTimer := addTimer(state.timers,state.timersLength,JUMPSCARE\_LENGTH);

**end**;

**end**;

//функции искусственного интеллекта монстров

//Монстр Джёрма

**function** jermaNextTimer(): real;

**begin**

result := getRandomInt(JERMA\_MIN\_OPP\_TIME, JERMA\_MAX\_OPP\_TIME);

**end**;

//Перемещение в следующую комнату

**function** jermaDirection(recentCamera, activeCamera: Cameras): Cameras;

**begin**

**var** res := -1;

**var** dangerChance := getRandomFloat(0, 1);

**if**(dangerChance < JERMA\_OFFICE\_CHANCE) **then begin**

**case**(activeCamera) **of**

CAMERA\_HALL: res := ord(CAMERA\_OFFICE\_RIGHT);

CAMERA\_CREEPY: res := ord(CAMERA\_OFFICE\_LEFT);

CAMERA\_CASINO, CAMERA\_SPIDER: res := ord(CAMERA\_VENT);

**end**;

**end**;

**if**(res <> -1) **then**

state.timers[state.jerma.timer] += 5 \* 120;

**while** (res = -1) **do**

**begin**

**var** randomCamera := getRandomInt(0, 8);

**var** available := JERMA\_DIRECTION\_MATRIX[ord(activeCamera)][randomCamera];

**if**(available = 1) **then**

**if**(randomCamera <> ord(recentCamera)) **and** (activeCamera <> CAMERA\_DARK) **or**

(getRandomFloat(0, 1) > JERMA\_RETURN\_CHANCE) **then**

res := randomCamera;

**end**;

Result := Cameras(res);

**end**;

**procedure** jermaAI;

**begin**

//удар электричеством

**if**(state.timers[state.shockTimer] >= 0) **and** (state.timers[state.shockTimer] - frameDelay <= 0) **and**

(state.jerma.camera = state.shock) **then**

**begin**

state.jerma.recentCamera := CAMERA\_SOLAR;

state.jerma.camera := CAMERA\_SOLAR;

state.timers[state.jerma.timer] := jermaNextTimer();

**end**;

**if**(state.timers[state.shockTimer] >= 0) **then**

state.timers[state.jerma.timer] += frameDelay;

//Монстр Джёрма передвигается

**if**(state.timers[state.jerma.timer] <= 0) **then begin**

**if**((state.jerma.camera = CAMERA\_OFFICE\_RIGHT) **or** (state.jerma.camera = CAMERA\_OFFICE\_LEFT) **or** (state.jerma.camera = CAMERA\_VENT)) **then**

**begin**

initiateJumpscare(state.jumpscare, JUMPSCARE\_JERMA, state.jumpscareTimer);

state.timers[state.jerma.timer] := 9999;

**end**

**else begin**

**var** randomLvl := getRandomInt(1, 20);

state.timers[state.jerma.timer] := jermaNextTimer();

**if**(randomLvl <= state.jerma.lvl) **then**

**begin**

**var** cameraSave := state.jerma.camera;

state.jerma.camera := jermaDirection(state.jerma.recentCamera, state.jerma.camera);

state.jerma.recentCamera := cameraSave;

**end**;

**end**;

**end**;

**end**;

**end**.

{$F+}

//Модуль "Визуал" для визуальных эффектов

**unit** visuals;

**interface**

**uses** saveState, drawing, math, timer;

//таймер для анимаций перехода

**var**

transition: **record**

timer: integer;

duration: real;

proc: **procedure**();

**end**;

//переменная длительности кадра в миллисекундах

recentFrame: real;

//относительная задержка по сравнением с эталонным временем кадра

frameDelay: real;

//помехи на экране

//пустая процедура

**procedure** nothing();

//создание помех

**procedure** makeNoise(minTime, maxTime, minVal, maxVal: real);

//прорисовка помех

**procedure** updateNoise();

**procedure** drawStatic(camera: CameraType; x,y:real);

//процедуры плавного перехода

**procedure** startTransition(duration: real; proc: **procedure**());

**procedure** updateTransition();

**implementation**

**procedure** nothing;

**begin**

**end**;

**procedure** makeNoise;

**begin**

state.timers[state.noise.timer] := getRandomFloat(minTime, maxTime);

state.noise.posY := getRandomFloat(-W\_HEIGHT \* 0.5, W\_HEIGHT \* 0.5);

state.noise.sizeY := getRandomFloat(minVal, maxVal);

**end**;

//прорисовка помех

**procedure** updateNoise;

**begin**

**if**(state.timers[state.noise.timer] > 0) **then begin**

drawRect(state.camera, state.camera.x,state.camera.y + state.noise.posY, W\_WIDTH, state.noise.sizeY, RGB(255, 255, 255));

**end**;

**end**;

**procedure** drawStatic;

**begin**

//прорисовка помех

drawSprite(state.camera, 'static/' + getRandomInt(1, 8) + '.png', x,y, W\_WIDTH,W\_HEIGHT);

updateNoise();

**end**;

//процедуры плавного перехода

**procedure** startTransition;

**begin**

**if**(state.timers[transition.timer] > -duration \* 0.5) **then begin**

**if**(state.timers[transition.timer] < 0) **then**

state.timers[transition.timer] := -state.timers[transition.timer];

**end**

**else**

state.timers[transition.timer] := duration \* 0.5;

transition.duration := duration;

transition.proc := proc;

**end**;

**procedure** updateTransition;

**begin**

**var** transitionLvl := clamp(1 - abs(state.timers[transition.timer]) / (transition.duration \* 0.5), 0, 1);

drawRect(state.camera, state.camera.x,state.camera.y, W\_WIDTH, W\_HEIGHT, ARGB(round(transitionLvl \* 255), 0, 0, 0));

**if**(state.timers[transition.timer] >= 0) **and** (state.timers[transition.timer] - frameDelay <= 0) **then**

transition.proc();

**end**;

**begin**

transition.timer := addTimer(state.timers,state.timersLength,0);

**end**.

//Модуль "Сохранения" для сохранения состояния игры

**unit** saves;

**interface**

**uses** saveState;

//процедуры работы с файлом сохранения

**procedure** loadSave();

**procedure** makeSave();

**implementation**

//процедуры работы с файлом сохранения

**procedure** loadSave;

**begin**

**var** f: **file**;

reset(f,'save.txt');

read(f, state);

close(f);

**end**;

**procedure** makeSave;

**begin**

**var** f : **file**;

reset(f,'save.txt');

**if**(state.gameState = STATE\_GAME) **and** (state.gameState = STATE\_PAUSE) **and**

(state.timers[state.jumpscareTimer]<0) **then**

**begin**

state.gameState := STATE\_GAME;

rewrite(f);

write(f, state);

**end**;

close(f);

**end**;

**end**.

**unit** game;

**interface**

**uses** graphWPF, drawing, saveState, input, AI, math, buttons, visuals, saves, timer;

**procedure** cycle();

**implementation**

**const**

//кол-во камер на карте

CAMERA\_COUNT = 9;

//длительность анимации поднятия планшета

LAPTOP\_ANIMATION\_LENGTH = FPS / 12;

//продолжительность игры

GAME\_LENGTH = FPS \* 60 \* 2.25;

//константы для удара электричеством

SHOCK\_DURATION = 120;

SHOCK\_EFFECT = 60;

SHOCK\_RELOAD = 10 \* FPS;

//длительность концовки

ENDING\_LENGTH = 3 \* FPS;

FLASH\_FREQUENCY = 60;

//для просмотра по сторонам

INACTIVE\_ZONE = 80;

TURN\_SPEED = 0.04;

**var**

//объекты камер на карте

cameraObjects: **array** [0..CAMERA\_COUNT-1] **of** MapCameraObjType;

**procedure** changeState(**var** camera: CameraType; **var** inGameState: InGameStateType; neededState: InGameStateType);

**begin**

**case** (neededState) **of**

STATE\_OFFICE:

**if**(inGameState = STATE\_BACK) **then**

**if**(mouse.x > 0) **then begin**

**var** neededX := -W\_WIDTH \* 2.25;

camera.x := neededX - (camera.targetX - camera.x);

camera.targetX := neededX;

**end**

**else begin**

**var** neededX := W\_WIDTH \* 2.25;

camera.x := neededX - (camera.targetX - camera.x);

camera.targetX := neededX;

**end**

**else**

**begin**

camera.targetX := 0;

camera.targetY := 0;

**end**;

STATE\_BACK:

**begin**

**if**(mouse.x > 0) **then**

camera.x := camera.x - 4.5 \* W\_WIDTH;

camera.targetX := -W\_WIDTH \* 2.25;

**end**;

STATE\_VENT:

**begin**

camera.targetY := -W\_HEIGHT \* 2;

camera.targetX := 0;

**end**;

**end**;

inGameState := neededState;

**end**;

**procedure** clearState();

**begin**

//оставляем место для таймеров плавного перехода и шумов

state.timersLength := 2;

state.camera.x := 0;

state.camera.y := 0;

state.camera.targetX := 0;

state.camera.targetY := 0;

state.inGameState := STATE\_OFFICE;

state.gameState := STATE\_GAME;

//начальная камера

state.currentCamera := ord(CAMERA\_SOLAR);

//Начальные параметры монстра Джёрма

state.jerma.camera := CAMERA\_SOLAR;

state.jerma.recentCamera := CAMERA\_DARK;

state.jerma.timer := addTimer(state.timers, state.timersLength, FPS \* getRandomFloat(1, 5));

state.jerma.lvl := 13;

state.jumpscare := JUMPSCARE\_NONE;

state.tabletAnimationFrames := 0;

state.shockTimer := addTimer(state.timers, state.timersLength, -1);

state.shockReloadTimer := addTimer(state.timers, state.timersLength, -1);

state.gameTimer := addTimer(state.timers, state.timersLength, GAME\_LENGTH);

**end**;

**procedure** updateLaptop(additionalOffset: real);

**begin**

**var** tabletY := (W\_HEIGHT + 25) \* (1 - state.tabletAnimationFrames / LAPTOP\_ANIMATION\_LENGTH);

**var** cameraSrc := 'cams/cam' + (state.currentCamera + 1) + '.png';

**var** camInGameWidth := W\_HEIGHT / ImageHeight('bitmaps/' + cameraSrc) \* ImageWidth('bitmaps/' + cameraSrc);

state.cameraOffset := clamp(state.cameraOffset, (-camInGameWidth + W\_WIDTH) \* 0.5, (camInGameWidth - W\_WIDTH) \* 0.5);

//прорисовка того, что видно на камере

//фон

drawSprite(state.camera, cameraSrc, state.camera.x - state.cameraOffset, state.camera.y + tabletY, camInGameWidth, W\_HEIGHT);

//прорисовка Джёрмы

**if**(state.currentCamera = ord(state.jerma.camera)) **then**

drawSprite(state.camera, 'jerma/cam' + (state.currentCamera + 1) + '.png', state.camera.x - state.cameraOffset, state.camera.y + tabletY, camInGameWidth, W\_HEIGHT);

//прорисовка карты камер

**var** mapX := state.camera.x + W\_WIDTH \* 0.5 - W\_HEIGHT \* 0.36;

**var** mapY := state.camera.y + W\_HEIGHT \* 0.16 + tabletY;

drawSprite(state.camera, 'map.png', mapX, tabletY + mapY, W\_HEIGHT \* 0.66, W\_HEIGHT \* 0.66);

**for var** cameraIndex := 0 **to** CAMERA\_COUNT - 1 **do**

**begin**

**var** mapCamera := cameraObjects[cameraIndex];

**var** cameraPosX := mapX + mapCamera.x;

**var** cameraPosY := mapY + mapCamera.y;

**if**(checkButtonZone(cameraPosX, cameraPosY, mapCamera.width, mapCamera.height, mouse.gamePosX, mouse.gamePosY, true) **and**

(mapCamera.cType = CAMERA\_TYPE\_RECT) **or** (mapCamera.cType = CAMERA\_TYPE\_ROUND) **and**

(distanceBetweenPoints(mouse.gamePosX, mouse.gamePosY, cameraPosX, cameraPosY) < mapCamera.width \* 0.58)) **then**

**begin**

mapCamera.color := ARGB(255, 255, 0, 0);

**if**(mouse.wentDown) **then begin**

makeNoise(4, 8, 100, 300);

state.currentCamera := cameraIndex;

state.cameraOffset := 0;

mouse.isDown := false;

**end**;

**end**

**else** mapCamera.color := ARGB(255, 0, 0, 0);

**if**(cameraIndex = state.currentCamera) **then** mapCamera.color := ARGB(255, 0, 255, 0);

**case** (mapCamera.cType) **of**

CAMERA\_TYPE\_RECT:

**begin**

drawRect(state.camera, cameraPosX, cameraPosY + tabletY, mapCamera.width, mapCamera.height, mapCamera.color);

drawRect(state.camera, cameraPosX, cameraPosY + tabletY, mapCamera.width - 15, mapCamera.height - 15, RGB(255, 255, 255));

**end**;

CAMERA\_TYPE\_ROUND:

**begin**

drawCircle(state.camera, cameraPosX, cameraPosY + tabletY, mapCamera.width \* 0.5, mapCamera.color);

drawCircle(state.camera, cameraPosX, cameraPosY + tabletY, mapCamera.width \* 0.5 - 7.5, RGB(255, 255, 255));

**end**;

**end**;

**end**;

drawStatic(state.camera, state.camera.x, state.camera.y + tabletY);

//анимация поднятия планшета и его прорисовка

drawSprite(state.camera, 'camera.png', state.camera.x, state.camera.y + tabletY, W\_WIDTH, W\_HEIGHT);

//движение камеры

**if**(mouse.isDown) **then**

state.cameraOffset += additionalOffset;

**end**;

**procedure** updateMenu();

**begin**

state.camera.x := 0;

state.camera.y := 0;

drawSprite(state.camera, 'menu.jpg', 400, 0, W\_HEIGHT / 3 \* 4, W\_HEIGHT);

drawText(state.camera, 0, -400, W\_WIDTH - 300, 80, 'One night at Jerma`s', RGB(255, 255, 255), 'True Lies', LeftCenter);

**if**(checkPushButton(-600, -250, 300, 50, mouse.x, mouse.y, 'Start game', mouse.isDown, FPS, frameDelay, LeftCenter)) **then**

startTransition(FPS, **procedure** -> state.gameState := STATE\_BEFORE\_NIGHT);

**var** f: **file**;

reset(f, 'save.txt');

**if**(f.Size > 0) **then begin**

**if**(checkPushButton(-600, -100, 300, 50, mouse.x, mouse.y, 'Continue', mouse.isDown, FPS, frameDelay, LeftCenter)) **then**

startTransition(FPS, **procedure** -> loadSave());

**end**

**else**

drawText(state.camera, -600, -100, 300, 50, 'Continue', ARGB(127, 255, 255, 255), 'True Lies', LeftCenter);

close(f);

**if**(checkPushButton(-600, 350, 300, 50, mouse.x, mouse.y, 'Exit', mouse.isDown, FPS, frameDelay, LeftCenter)) **then**

startTransition(FPS \* 2, **procedure** -> window.close());

**end**;

**procedure** updateWarning();

**begin**

drawText(state.camera, 0, -200, W\_WIDTH, 100, 'Warning', RGB(255, 255, 255));

drawText(state.camera, 0, 200, W\_WIDTH, 50, 'This game has loud noises and jumpscares', RGB(255, 255, 255));

**if**(mouse.wentDown) **then**

startTransition(FPS, **procedure** -> state.gameState := STATE\_MENU);

drawText(state.camera, -W\_WIDTH \* 0.5 + 180, W\_HEIGHT\*0.5 - 50,1,30,'One night at Jerma`s');

drawText(state.camera, W\_WIDTH \* 0.5 - 180, W\_HEIGHT\*0.5 - 50,1,30,'Алексей Рогатин 1413');

**end**;

**procedure** updateBeforeNight();

**begin**

drawParagraph(state.camera, state.camera.x, state.camera.y - 400, W\_WIDTH \* 0.42, 50, 2,

'You are a fan of a very famous internet celebrity Jerma, who became famous because of his show "Peep the horror". One night you broke into his house to get his autograph. It turned out that the Horror was a real l.iving being. After that Jerma is no longer willing to keep you alive',

RGB(255, 255, 255), 'True Lies', center);

**if**(mouse.wentDown) **then**

startTransition(FPS, **procedure** -> clearState());

**end**;

**procedure** updateGameOver();

**begin**

drawText(state.camera, 0, 0, 0, 120, 'You died', RGB(240, 0, 0));

**if**(mouse.wentDown) **then**

startTransition(FPS, **procedure** -> state.gameState := STATE\_MENU);

**end**;

**procedure** updateWin();

**begin**

state.camera.x := 0;

state.camera.y := 0;

**if**(state.timers[transition.timer] > 0) **or** (state.timers[transition.timer] < -ENDING\_LENGTH)

**or** ((abs(floor(state.timers[transition.timer])) **mod** FLASH\_FREQUENCY) <= FLASH\_FREQUENCY / 2) **then**

drawText(state.camera, 0, 0, 0, 120, '6 AM', RGB(255, 255, 255), 'Impact');

**if**(mouse.wentDown) **then**

startTransition(FPS, **procedure** -> state.gameState := STATE\_MENU);

**end**;

**procedure** updatePause();

**begin**

**if**(escKey.wentDown) **then**

state.gameState := STATE\_GAME;

drawText(state.camera, state.camera.x, state.camera.y - 300, 0, 80, 'Pause', RGB(255, 255, 255));

**if**(checkPushButton(state.camera.x, state.camera.y, 400, 80, mouse.gamePosX, mouse.gamePosY, 'Save & exit',

mouse.isDown, FPS, frameDelay, center)) **then**

**begin**

makeSave();

startTransition(FPS, **procedure** -> state.gameState := STATE\_MENU);

**end**;

**end**;

**procedure** updateGame();

**begin**

//корридоры

drawSprite(state.camera, 'behindTheDoorsLeft.png', -W\_WIDTH \* 0.53, 0, 0.18 \* W\_WIDTH, W\_HEIGHT);

drawSprite(state.camera, 'behindTheDoorsRight.png', W\_WIDTH \* 0.53, 0, 0.18 \* W\_WIDTH, W\_HEIGHT);

drawSprite(state.camera, 'behindTheVent.png', 0, -W\_HEIGHT \* 2.03, W\_WIDTH \* 0.45, W\_WIDTH \* 0.45);

//эффект электрического разряда

**var** shockLvl := clamp(state.timers[state.shockTimer] / SHOCK\_EFFECT, 0, 1) \* integer(state.timers[state.shockTimer] <= SHOCK\_EFFECT) \* 255;

drawRect(state.camera, -W\_WIDTH \* 0.53, 0, 0.18 \* W\_WIDTH, W\_HEIGHT,

ARGB(round(255 - shockLvl \* integer(state.shock = CAMERA\_OFFICE\_LEFT)), 0, 0, 0));

drawRect(state.camera, W\_WIDTH \* 0.53, 0, 0.18 \* W\_WIDTH, W\_HEIGHT,

ARGB(round(255 - shockLvl \* integer(state.shock = CAMERA\_OFFICE\_RIGHT)), 0, 0, 0));

drawRect(state.camera, 0, -W\_HEIGHT \* 2.03, W\_WIDTH \* 0.45, W\_WIDTH \* 0.45,

ARGB(round(255 - shockLvl \* integer(state.shock = CAMERA\_VENT)), 0, 0, 0));

//перемещение камеры по сторонам

**var** additionalOffset := max((abs(mouse.x) - 60) \* 0.04, 0);

additionalOffset \*= frameDelay;

**if**(mouse.x < 0) **then** additionalOffset \*= -1;

**if**(state.timers[state.shockTimer] > 0)**and**(state.shock = state.jerma.camera) **and**

(state.timers[state.shockTimer] < SHOCK\_EFFECT)**and**(state.shock = CAMERA\_VENT) **then**

drawSprite(state.camera, 'jermaVent.png', 0, -W\_HEIGHT \* 2.03, W\_WIDTH \* 0.5, W\_WIDTH \* 0.5);

drawSprite(state.camera, 'newOffice.png', 0, -W\_HEIGHT, W\_WIDTH \* 11, W\_HEIGHT \* 3);

//Джёрма в офисе

**if**(state.timers[state.shockTimer] > 0)**and**(state.shock = state.jerma.camera) **and**

(state.timers[state.shockTimer] < SHOCK\_EFFECT) **then begin**

**case** state.shock **of**

CAMERA\_OFFICE\_RIGHT:drawSprite(state.camera, 'jermaRight.png', W\_WIDTH \* 0.53, 0, 0.18 \* W\_WIDTH, W\_HEIGHT);

CAMERA\_OFFICE\_LEFT:drawSprite(state.camera, 'jermaLeft.png', -W\_WIDTH \* 0.53, 0, 0.18 \* W\_WIDTH, W\_HEIGHT);

**end**;

**end**;

//анимация опускания планшета

**if**(state.inGameState <> STATE\_LAPTOP) **then begin**

state.tabletAnimationFrames := max(0, state.tabletAnimationFrames - frameDelay);

**end**

**else begin**

state.tabletAnimationFrames := min(LAPTOP\_ANIMATION\_LENGTH, state.tabletAnimationFrames + frameDelay);

**end**;

**if**(state.tabletAnimationFrames > 0) **then begin**

updateLaptop(additionalOffset);

**end**;

**case**(state.inGameState) **of**

STATE\_OFFICE:

**begin**

//перемещение камеры вправо и влево

**var** cameraLimit := W\_WIDTH \* 0.25;

state.camera.targetX += additionalOffset;

state.camera.targetX := clamp(state.camera.targetX, -cameraLimit, cameraLimit);

//кнопки для офиса

**if**(checkActiveZone(0, W\_HEIGHT \* 0.5 - 25, 1400, 50, mouse)) **then begin**

changeState(state.camera, state.inGameState, STATE\_LAPTOP);

**end**;

**if**(checkActiveZone(0, -W\_HEIGHT \* 0.5 + 25, 1400, 50, mouse)) **then**

changeState(state.camera, state.inGameState, STATE\_VENT);

**if**(state.camera.targetX = cameraLimit) **then begin**

drawSprite(state.camera, 'rightButton.png', state.camera.x + W\_WIDTH \* 0.5 - 25, state.camera.y, 50, 500);

**if**(checkActiveZone(W\_WIDTH \* 0.5 - 25, 0, 50, 500, mouse)) **then**

changeState(state.camera, state.inGameState, STATE\_BACK);

**end**;

**if**(state.camera.targetX = -cameraLimit) **then begin**

drawSprite(state.camera, 'leftButton.png', state.camera.x - (W\_WIDTH \* 0.5 - 25), state.camera.y, 50, 500);

**if**(checkActiveZone(-W\_WIDTH \* 0.5 + 25, 0, 50, 500, mouse)) **then**

changeState(state.camera, state.inGameState, STATE\_BACK);

**end**;

//нажатие на двери

**if**(state.timers[state.shockReloadTimer] < 0) **then begin**

**if**(checkButtonZone(-W\_WIDTH \* 0.52, 0, 0.36 \* W\_WIDTH, W\_HEIGHT, mouse.gamePosX, mouse.gamePosY, true)) **then begin**

**if**(mouse.wentDown) **then begin**

state.timers[state.shockTimer] := SHOCK\_DURATION;

state.shock := CAMERA\_OFFICE\_LEFT;

state.timers[state.shockReloadTimer] := SHOCK\_RELOAD;

**end**;

**end**;

**if**(checkButtonZone(W\_WIDTH \* 0.52, 0, 0.36 \* W\_WIDTH, W\_HEIGHT, mouse.gamePosX, mouse.gamePosY, true)) **then begin**

**if**(mouse.wentDown) **then begin**

state.timers[state.shockTimer] := SHOCK\_DURATION;

state.shock := CAMERA\_OFFICE\_RIGHT;

state.timers[state.shockReloadTimer] := SHOCK\_RELOAD;

**end**;

**end**;

**end**;

drawSprite(state.camera, 'invertedButton.png', state.camera.x, state.camera.y - W\_HEIGHT \* 0.5 + 25, 1400, 50);

drawSprite(state.camera, 'button.png', state.camera.x, state.camera.y + W\_HEIGHT \* 0.5 - 25, 1400, 50);

**end**;

STATE\_VENT:

**begin**

**if**(checkActiveZone(0, W\_HEIGHT \* 0.5 - 25, 1400, 50, mouse)) **then**

changeState(state.camera, state.inGameState, STATE\_OFFICE);

//нажатие на вентиляцию

**if**(state.timers[state.shockReloadTimer] < 0) **then begin**

**if**(checkButtonZone(0, -W\_HEIGHT \* 2.03, W\_WIDTH \* 0.45, W\_WIDTH \* 0.45, mouse.gamePosX, mouse.gamePosY, true)) **then begin**

**if**(mouse.wentDown) **then begin**

state.timers[state.shockTimer] := SHOCK\_DURATION;

state.shock := CAMERA\_VENT;

state.timers[state.shockReloadTimer] := SHOCK\_RELOAD;

**end**;

**end**;

**end**;

drawSprite(state.camera, 'button.png', state.camera.x, state.camera.y + W\_HEIGHT \* 0.5 - 25, 1400, 50);

**end**;

STATE\_BACK:

**begin**

drawSprite(state.camera, 'rightButton.png', state.camera.x + W\_WIDTH \* 0.5 - 25, state.camera.y, 50, 500);

drawSprite(state.camera, 'leftButton.png', state.camera.x - (W\_WIDTH \* 0.5 - 25), state.camera.y, 50, 500);

**if**(checkActiveZone(W\_WIDTH \* 0.5 - 25, 0, 50, 500, mouse) **or**

checkActiveZone(-W\_WIDTH \* 0.5 + 25, 0, 50, 500, mouse)) **then**

changeState(state.camera, state.inGameState, STATE\_OFFICE);

**end**;

STATE\_LAPTOP:

**begin**

//кнопка возврата для планшета

**if**(checkActiveZone(0, W\_HEIGHT \* 0.5 - 25, 1400, 50, mouse)) **then begin**

state.inGameState := STATE\_OFFICE;

**end**;

drawSprite(state.camera, 'button.png', state.camera.x, state.camera.y + W\_HEIGHT \* 0.5 - 25, 1400, 50);

**end**;

**end**;

**if**(escKey.wentDown) **then**

state.gameState := STATE\_PAUSE;

**if**(state.timers[state.gameTimer] < 0) **then**

startTransition(120, **procedure** -> state.gameState := STATE\_WIN);

//реализация поведения монстра Джёрма

jermaAI();

//скример и окончание игры

**if**(state.jumpscare <> JUMPSCARE\_NONE) **and** (state.timers[state.gameTimer] > 0) **then begin**

changeState(state.camera, state.inGameState, STATE\_OFFICE);

state.camera.x := state.camera.targetX;

state.camera.y := state.camera.targetY;

state.tabletAnimationFrames := 0;

**if**(state.timers[state.jumpscareTimer] <= 0) **then begin**

**var** f: **file**;

reset(f, 'save.txt');

rewrite(f);

close(f);

state.gameState := STATE\_GAME\_OVER;

**end**

**else**

drawSprite(state.camera,'jermaScreamer/'+(20-round(state.timers[state.jumpscareTimer]/JUMPSCARE\_LENGTH\*19))+'.jpg',

state.camera.x,state.camera.y,W\_WIDTH,W\_HEIGHT);

**end**;

//отображение полосы заряда шока

**var** shockReloadLvl := 1 - clamp(state.timers[state.shockReloadTimer] / SHOCK\_RELOAD, 0, 1);

**var** barHeight := 150 \* shockReloadLvl;

drawSprite(state.camera, 'shockLvl.png', state.camera.x - W\_WIDTH \* 0.5 + 100, state.camera.y - W\_HEIGHT \* 0.5 + 150, 50, 150);

drawRect(state.camera, state.camera.x - W\_WIDTH \* 0.5 + 100, state.camera.y - W\_HEIGHT \* 0.5 + 150 + (150 - barHeight) \* 0.5, 50, barHeight, ARGB(round(55 + 100 \* power(shockReloadLvl, 3)), 0, 0, 255));

//отображение времени

**var** time := (GAME\_LENGTH - state.timers[state.gameTimer]) / GAME\_LENGTH \* 6;

**var** hours := floor(time);

**var** minutes := '' + floor((time - hours) \* 60);

**if**(floor((time - hours) \* 60) < 10) **then**

minutes := '0' + minutes;

drawText(state.camera, state.camera.x + W\_WIDTH \* 0.5 - 150, state.camera.y - W\_HEIGHT \* 0.5 + 100, 0, 70, hours + ':' + minutes, RGB(10, 10, 255), 'Impact', center);

**end**;

**procedure** cycle;

**begin**

window.CenterOnScreen();

window.Clear(RGB(0, 0, 0));

//нахождение задержки с прошлого кадра

recentFrame := getTime() - recentFrame;

recentFrame := max(0, recentFrame);

frameDelay := recentFrame / (1000 / FPS) / 2;

recentFrame := getTime();

//нахождение координат мышки в мире

mouse.gamePosX := mouse.x + state.camera.x;

mouse.gamePosY := mouse.y + state.camera.y;

**if**(**not** mouse.isDown) **then**

btnPushing := 0;

//выбор в зависимости от состояния игры

**case** (state.gameState) **of**

STATE\_WARNING: updateWarning();

STATE\_MENU: updateMenu();

STATE\_BEFORE\_NIGHT: updateBeforeNight();

STATE\_GAME\_OVER: updateGameOver();

STATE\_PAUSE: updatePause();

STATE\_WIN: updateWin();

STATE\_GAME: updateGame();

**end**;

**if**(state.gameState <> STATE\_GAME) **and** (state.gameState <> STATE\_WIN) **then**

**begin**

drawStatic(state.camera, state.camera.x, state.camera.y);

**if**(state.timers[state.noise.timer] <= 0) **and** (getRandomFloat(0, 1) > 0.99) **then**

makeNoise(2, 8, 50, 150);

**end**;

//очистка ввода

clearInput();

//сохранение прошлых значений координат мышки

mouse.recentPosX := mouse.x;

mouse.recentPosY := mouse.y;

**if**(state.gameState <> STATE\_PAUSE) **then**

**begin**

updateTimers(state.timers, frameDelay, state.timersLength);

//перемещение камеры

state.camera.x += (state.camera.targetX - state.camera.x) \* CAMERA\_SPEED \* frameDelay;

state.camera.y += (state.camera.targetY - state.camera.y) \* CAMERA\_SPEED \* frameDelay;

**end**

**else**

updateTimers(state.timers, frameDelay, 2);

updateTransition();

**end**;

**begin**

//подготовка камер к работе

//основные свойства камер на карте камер

**for var** i := 0 **to** CAMERA\_COUNT - 1 **do**

**begin**

cameraObjects[i].cType := CAMERA\_TYPE\_RECT;

cameraObjects[i].width := 120;

cameraObjects[i].height := 90;

**end**;

cameraObjects[ord(CAMERA\_SOLAR)].cType := CAMERA\_TYPE\_ROUND;

//пространсвенные свойства для каждой камеры

cameraObjects[ord(CAMERA\_SOLAR)].width := 105;

cameraObjects[ord(CAMERA\_HALL)].x := 75;

cameraObjects[ord(CAMERA\_HALL)].y := 150;

cameraObjects[ord(CAMERA\_HALL)].width := 60;

cameraObjects[ord(CAMERA\_HALL)].height := 180;

cameraObjects[ord(CAMERA\_CREEPY)].x := -75;

cameraObjects[ord(CAMERA\_CREEPY)].y := 150;

cameraObjects[ord(CAMERA\_CREEPY)].width := 60;

cameraObjects[ord(CAMERA\_CREEPY)].height := 180;

cameraObjects[ord(CAMERA\_DARK)].x := -165;

cameraObjects[ord(CAMERA\_DARK)].y := 150;

cameraObjects[ord(CAMERA\_DARK)].width := 60;

cameraObjects[ord(CAMERA\_DARK)].height := 60;

cameraObjects[ord(CAMERA\_RING)].x := 0;

cameraObjects[ord(CAMERA\_RING)].y := -150;

cameraObjects[ord(CAMERA\_CASINO)].x := -150;

cameraObjects[ord(CAMERA\_CASINO)].y := -105;

cameraObjects[ord(CAMERA\_SPIDER)].x := 210;

cameraObjects[ord(CAMERA\_SPIDER)].y := -150;

cameraObjects[ord(CAMERA\_CAGE)].x := 210;

cameraObjects[ord(CAMERA\_CAGE)].y := 0;

cameraObjects[ord(CAMERA\_BATH)].x := 210;

cameraObjects[ord(CAMERA\_BATH)].y := 150;

**end**.