

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
«Вычислительная техника»
(наименование)

Кафедра
«Системы автоматизированного проектирования»
(наименование)

Направление подготовки

09.03.01 «Информатика и вычислительная техника»
(код и наименование)

Профиль

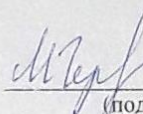
«Системы автоматизированного проектирования»
(наименование)

БАКАЛАВРСКАЯ РАБОТА

на тему

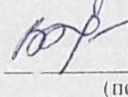
Разработка системы автоматической индикации игровых ситуаций на примере RPG-игр.

Студент

 29.05.2017
(подпись, дата)

Черепашук М.М.
(фамилия, инициалы)

Руководитель

 29.05.2017
(подпись, дата)

Евн В.В.
(фамилия, инициалы)

Нормоконтролёр

проф. каф. САПР ПГУ
(должность, место работы)

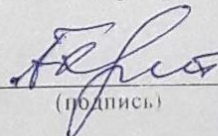
 2.05.17
(подпись, дата)

Бондарь А.С.
(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от

№

Заведующий кафедрой


(подпись)

Варшавский А.М.
(фамилия, инициалы)

Работа защищена с отметкой

(протокол заседания ГЭК от №)

Секретарь ГЭК

(подпись)

(фамилия, инициалы)

Пенза 2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	7
1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	9
1.1. Обзор аналогичных систем.....	9
1.2. Выбор языка программирования.....	10
1.3. Выводы по разделу	12
2. ПРОЕКТИРОВАНИЕ ИГРОВОЙ СИСТЕМЫ.....	13
2.1. Определение состава системы.....	13
2.2. Проектирование подсистемы развития героя.....	16
2.3. Проектирование подсистемы игрового мира	20
2.4. Проектирование подсистемы игровых состояний	22
2.5. Проектирование подсистемы повествования	23
2.6. Проектирование подсистемы игр-головоломок	26
2.7. Проектирование подсистемы автоматической генерации игровых ситуаций.....	28
2.8. Выводы по разделу	36
3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРОВОЙ СИСТЕМЫ	37
3.1. Программная реализация подсистемы развития героя	37
3.2. Программная реализация подсистемы игрового мира.....	38
3.3. Программная реализация подсистемы игровых состояний.....	40
3.4. Программная реализация подсистемы повествования	43
3.5. Программная реализация подсистемы игр-головоломок	44
3.6. Программная реализация подсистемы автоматической генерации игровых ситуаций.....	46
3.7. Контрольный пример	48
3.8. Выводы по разделу	52
ЗАКЛЮЧЕНИЕ	53
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	55
ПРИЛОЖЕНИЯ.....	57
ПРИЛОЖЕНИЕ А1	58

ПРИЛОЖЕНИЕ А2	59
ПРИЛОЖЕНИЕ А3	61
ПРИЛОЖЕНИЕ А4	64
ПРИЛОЖЕНИЕ А5	68
ПРИЛОЖЕНИЕ А6	71
ПРИЛОЖЕНИЕ Б1	74
ПРИЛОЖЕНИЕ Б2	76
ПРИЛОЖЕНИЕ Б3	78
ПРИЛОЖЕНИЕ Б4	80
ПРИЛОЖЕНИЕ Б5	84
ПРИЛОЖЕНИЕ Б6	86

ВВЕДЕНИЕ

Компьютерные игры или видеоигры – вид программного обеспечения, предназначенный для организации игрового процесса, связи с партнёрами по игре, или сама выступающая в качестве партнёра [1].

Компьютерные игры используются в различных целях: некоторые видят в видеоиграх лишь средство развлечения или отдыха, другие применяют их для развития интеллекта и логического мышления, а для некоторых компьютерные игры становятся настолько важной частью жизни, что заменяют реальный мир, становясь наркотиком. Также видеоигры разделяются по жанрам: от обычных стрелялок от первого лица до сложных симуляторов.

С точки зрения разработки игровых событий существует 2 основных подхода:

- 1) Создание игровых событий вручную;
- 2) Создание систем процедурной генерации событий.

При создании игровых событий вручную, программист самостоятельно задает начальные условия для каждого такого события, а также не ограничивает себя в разнообразии типов игровых событий и условий их выполнения. Например, сложные головоломки с использованием физических законов.

Так как такой способ требует много времени и трудовых ресурсов, то в основном используется в крупнобюджетных игровых проектах AAA - класса, целью которых, в подавляющем большинстве случаев, является повествование, а не игровой процесс.

В случае с использованием систем процедурной генерации события должны создаваться по определённым условиям, а также привязываются к определенным тематикам. Например, генерация случайных имен неигровых персонажей в играх, расположение некоторых игровых объектов в мире, генерация лабиринтов, генерация миров.

Вся сложность данного способа зависит лишь от функционала генератора, поэтому генерации игровых ситуаций чаще всего применяется в независимых проектах, так как удешевляют и ускоряют разработку.

На сегодняшний день все больше разработчиков ориентируются на использование систем генерации. В своем цикле статей [2] о процедурной генерации и ее будущем известный разработчик Марк Джонсон [3] описывает огромное число возможных способов применения систем генерации. Эти статьи и послужили толчком к определению темы выпускной квалификационной работы.

На основе изложенных данных была поставлена цель: разработка системы автоматической генерации игровых ситуаций на примере RPG-игр.

Исходя из поставленной цели, были сформулированы следующие задачи:

- 1) Исследование существующих аналогов, и выявление их достоинств и недостатков;
- 2) Проектирование основных игровых подсистем;
- 3) Разработка игры, для демонстрации спроектированных игровых подсистем;
- 4) Демонстрация контрольного примера.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Анализ предметной области – одна из первых задач, с которой сталкивается разработчик. Предметная область сильно влияет на все аспекты работы: требования к системе, взаимодействие с пользователем, реализацию.

1.1. Обзор аналогичных систем

В качестве рассматриваемых аналогов были выбраны следующие видеоигры:

- 1) Diablo III [4];
- 2) The Elder Scrolls V: Skyrim [5];
- 3) The Witness [6].

Данные видеоигры были выбраны на основе собственного опыта, который был приобретен в ходе их прохождения.

Diablo III

Diablo III – компьютерная игра в жанре Action/RPG, разработанная студией Blizzard Entertainment и выпущенная в 2012 году.

В этой игре реализована система генерации заданий путем выбора из общей базы заданий некоторого набора. На набор выбранных заданий оказывает влияние текущий уровень сложности.

The elder Scrolls V: Skyrim

The Elder Scrolls V: Skyrim – компьютерная игра в жанре RPG, разработанная студией Bethesda Game Studios и выпущенная в 2011 году.

В данной игре реализована система генерации заданий на очистку подземелий, где из базы точек входа в подземелье сопоставляется подземелье из базы подземелий.

The Witness

The Witness – компьютерная игра в жанре графического квеста, разработанная студией Thekla, Inc и выпущенная в 2016 году.

Принцип работы системы генерации: в игровом мире имеется набор объектов, при взаимодействии с которыми происходит запуск игры-

головоломки. Ее особенностью является то, что эти задания генерируются только после такого взаимодействия. Ниже представлен результат работы данной системы.

Для удобства сравнения результаты анализа систем-конкурентов занесены в таблицу 1.

Таблица 1 - Сравнение систем генерации.

Название	Diablo 3	The elder Scrolls V: Skyrim	The Witness
Принцип генерации игровых ситуаций	Выборка из набора	Сопоставление элементов баз	Генерация уровней
Наличие уровней сложности	13 уровней сложности	Нет	Нет
Независимость конкретных игровых ситуаций от конкретных точек входа	Нет	Частично	Да
Влияние ролевых элементов на генерацию игровых ситуаций	Частично	Нет	Нет

На основании полученных данных можно сформулировать проблему работы: «Как должна выглядеть игровая система, которая сочетала бы все вышепредложенные критерии?»

1.2. Выбор языка программирования

На сегодняшний день существует неисчислимое количество языков программирования. Они предназначены для решения различных задач, написание программ под различные аппаратные платформы. Для решения задач разработки игр наибольшей популярностью пользуются языки C++ [7], C# [8, 10-12] и Java [9].

Видеоигра достаточно сложная система. При ее разработке необходимо решать большое число задач:

- 1) Организация работы с аппаратной частью компьютера;
 - 2) Организация работы с различными видами ресурсов;
 - 3) Реализация сложных математических алгоритмов.
- и др.

Языки C++, C#, Java позволяют решать эти задачи. Для того, чтобы определить какой язык лучше использовать в данной работе, была проведена сравнительная характеристика этих языков по критериям, которые связаны с разрабатываемой игрой (таблица 2).

Таблица 2 - Сравнение языков программирования.

	C++	C#	Java
Простота в изучении языка	2	3	1
Удобство написания кода	2	3	1
Достаточность стандартных библиотек	1	3	1
Производительность программ	3	2	2
Итог	8	11	5

На основе полученной сравнительной характеристики было решено использовать язык C# для разработки игровой системы.

В качестве среды разработки используется Microsoft Visual Studio Microsoft Visual Studio 2013 так как данная версия поддерживает разработку приложений для .NET Framework 4.5.1.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня [13].

Также для экономии времени при разработке будет использоваться фреймворк MonoGame. Причины выбора:

- 1) Реализует внутри себя всю работу, связанную с взаимодействием программы и аппаратной части;
- 2) Является развитием Microsoft XNA (так как его поддержка была завершена в 2013) и обладает всеми его возможностями, в том числе комплексом реализованных механизмов, упрощающих разработку игр;
- 3) Активно поддерживается разработчиками;
- 4) Написание кода на языке C#;
- 5) Наличие документации [14].

После выбора языка программирования можно приступать непосредственно к созданию игровой системы.

1.3. Выводы по разделу

В ходе выполнения раздела были определены системы-конкуренты. Результат сравнения систем был продемонстрирован в виде таблицы. На основе сравнения была четко определена проблема для ее дальнейшего решения.

Также был проведен анализ языков программирования для реализации. По его результатам было решено использовать язык программирования C#, а для, того чтобы упрощения разработки базовых компонентов был использован фреймворк MonoGame.

2. ПРОЕКТИРОВАНИЕ ИГРОВОЙ СИСТЕМЫ

Проектирование программного обеспечения (ПО) - процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или её части. Результатом проектирования является проект — целостная совокупность моделей, свойств или характеристик, описанных в форме, пригодной для реализации системы [15].

В данной работе проектирование будет осуществляться в рамках объектно-ориентированной методологии (OOM), а все схемы будут создаваться на языке графического описания объектов Unified Modeling Language (UML) [16].

2.1. Определение состава системы

Для определения состава игровой системы необходимо ответить на следующие вопросы:

- 1) Какие действующие лица будет взаимодействовать с игровой системой?
- 2) Какие функции будут доступны этим лицам?

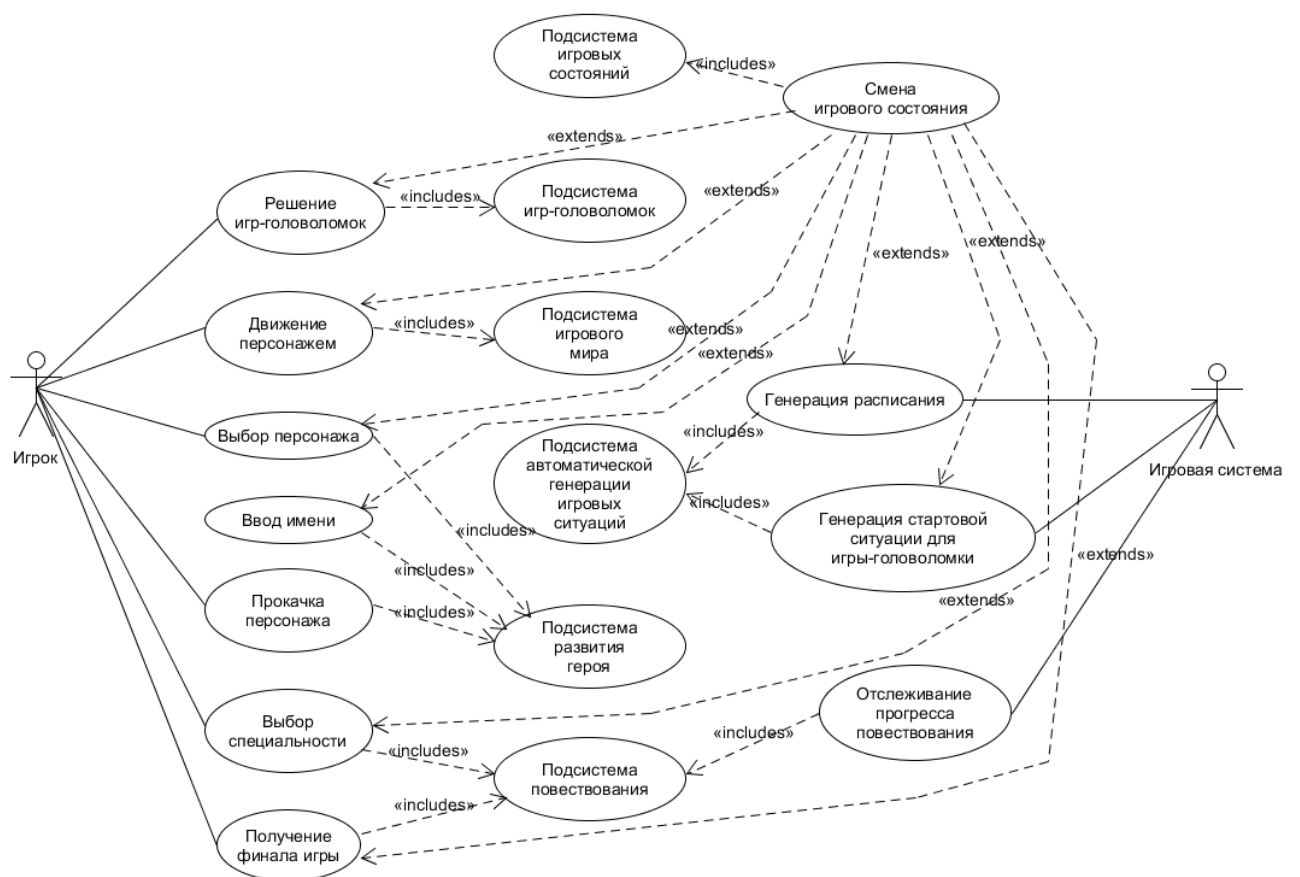
Чтобы ответить на эти вопросы была построена диаграмма прецедентов (рисунок 1).

Как видно из диаграммы прецедентов, было выделено 2 действующих лица:

- 1) Игрок – лицо, которое выполняет функции для прохождения игры;
- 2) Игровая системы – лицо, которое выполняет функции по формированию игрового окружения для игрока.

Также на диаграмме отчетливо выделяется набор подсистем, из которых состоит игровая система:

- 1) Подсистема игровых состояний – предназначена для управления игровыми состояниями и переходами между ними;



Каждая подсистема имеет набор входных и выходных данных для выполнения своих функций. Ниже приведена таблица с входными и выходными данными каждой подсистемы (таблица 3).

Таблица 3 - Входные и выходные данные подсистем.

Название подсистемы	Входные данные	Выходные данные
Подсистема развития героя	Пол персонажа; Имя персонажа; Специальность	Значения характеристик;
Подсистема повествования	Значения характеристик	Информация об игровом состоянии для подсистемы игровых состояний; Список актуальных предметов; Информация о позиции персонажа
Подсистема игрового мира	Информация о позиции персонажа; Информация об игровом мире;	Информация об игровом состоянии для подсистемы игровых состояний; Информация о точках входа
Подсистема игровых состояний	Информация об игровом состоянии;	Вывод на экран графическую информацию игрового состояния
Подсистема игра-головоломки	Информация об игре-головоломке	Информация об игровом состоянии для подсистемы игровых состояний
Подсистема автоматической генерации игровых ситуаций	Значения характеристик; Список актуальных предметов; Информация о точках входа	Набор пар «Точка входа» - «Игра-головоломка»

Как видно из таблицы некоторые выходные данные одних подсистем являются входными для других. Это связано с взаимодействием между подсистемами. На рисунке ниже представлена структурная схема взаимодействия подсистем (рисунок 2).

В следующих пунктах все эти подсистемы будут рассмотрены подробнее.

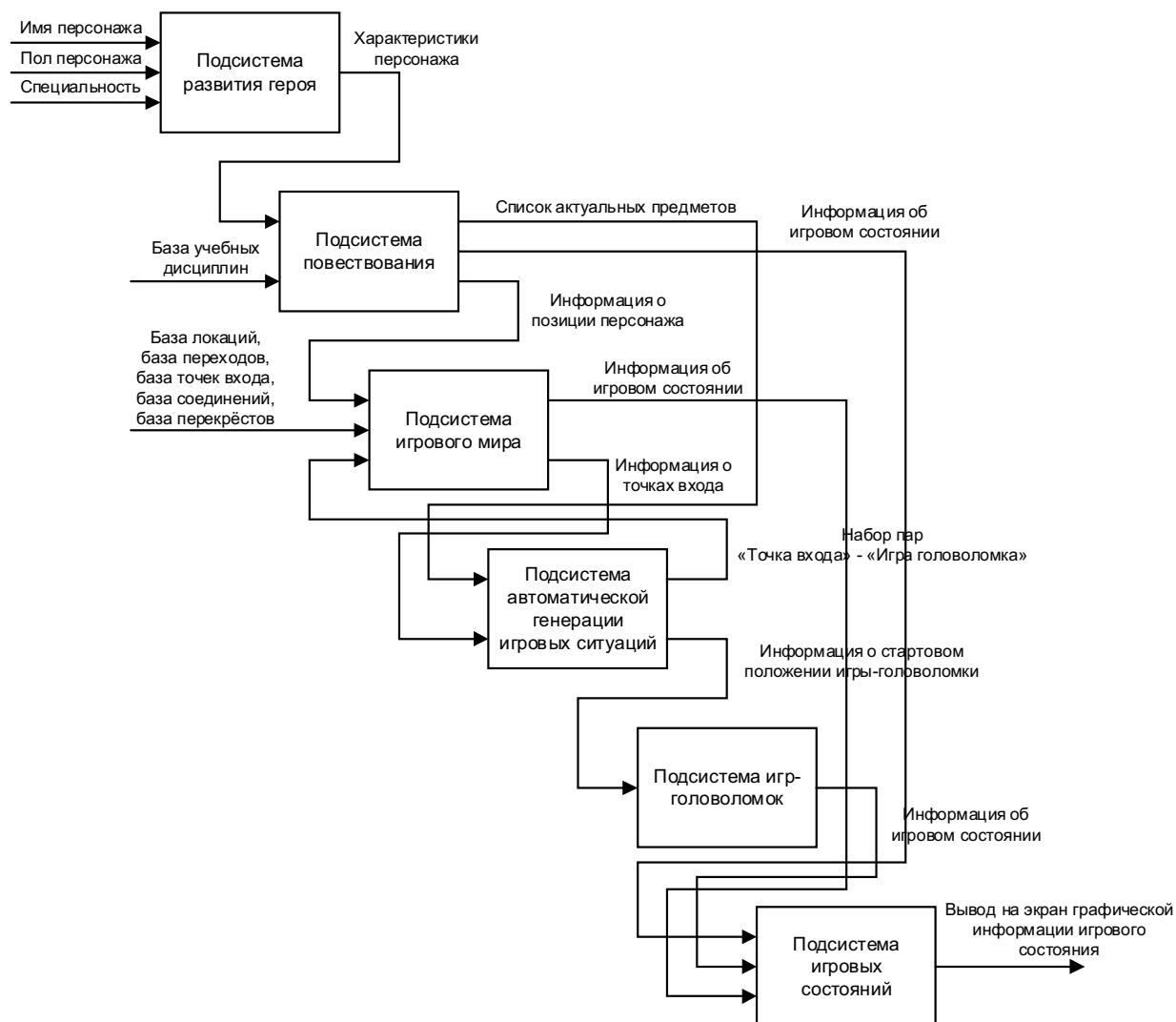


Рисунок 2 - Схема взаимодействия подсистем.

2.2. Проектирование подсистемы развития героя

Подсистема развития героя – представляет возможности управления характеристиками персонажа.

Функции, выполняемые с помощью подсистемы:

- 1) Выбор пола персонажа;
- 2) Выбор имени персонажа;
- 3) Выбор специальности
- 4) Управление характеристиками персонажа;

Первые 3 функции являются также необходимыми условиями для возможности использования 4 функции и, таким образом, полностью инициализируют подсистему.

Для того, чтобы описать, что подразумевает из себя 4 функция необходимо для начала определить основной набор основных и второстепенных характеристик, специальные показатели и навыков.

Основные характеристики – характеристики, на значения которых влияние оказывает сам игрок:

- 1) Выносливость;
- 2) Интеллект;
- 3) Красноречие;
- 4) Удача.

Второстепенные характеристики – характеристики, которые формируются из-за изменения основных характеристик:

- 1) Бонус опыта – коэффициент, определяющий количество дополнительно начисляемого опыта. Зависит от параметра «Интеллект»;
- 2) Бонус репутации – коэффициент, определяющий количество дополнительно начисляемой репутации. Зависит от параметра «Красноречие»;
- 3) Бонус денег – коэффициент, определяющий количество дополнительно начисляемых денег. Зависит от параметра «Красноречие»;
- 4) Шанс критического броска – обозначает вероятность с которой начисляемое количество опыта, репутации или денег может быть дополнительно увеличено. Зависит от параметра «Удача».
- 5) Бонус критического броска – коэффициент, определяющий количество дополнительно получаемого опыта, репутации или денег при выпадении критического броска. Зависит от параметра «Удача».

Специальные показатели – особые характеристики, на значения которых может оказывать как игрок, так и сама игровая система с помощью специальных функций:

- 1) Здоровье – показатель, характеризующий то, насколько персонаж близок к поражению. Определяется 2 составляющими: текущим значением и максимальным. Если значение равно 0 – персонаж умирает. Значение зависит от характеристики «Выносливость».
- 2) Уровень – показатель развития персонажа, его профессионального или иного мастерства, общей силы и значимости. Развитие персонажа через получения уровней — появившаяся первой система развития персонажа в настольных ролевых играх, остающаяся во многом актуальной до сих пор. За повышение уровня на 1 начисляется 1 очко характеристик.
- 3) Очки характеристик – показатель, определяющий то, насколько можно развить основные характеристики. Определяется 2 составляющими: количеством нераспределенных очков и общим числом полученных очков.
- 4) Опыт – показатель, характеризующий то, насколько персонаж близок к получению нового уровня. Определяется 2 составляющими: текущим значением опыта и количеством опыта, необходимого для получения нового уровня. Если значение текущего количества опыта достигает значения требуемого количества – происходит получение уровня, задание нового порога опыта и сброс значения текущего количества опыта к значению оставшемуся после вычета количества опыта, требуемого за предыдущий уровень.
- 5) Репутация – показатель, характеризующий то, насколько персонаж близок к открытию какого-либо навыка.
- 6) Деньги – показатель, служащий измерителем стоимости других благ, также используемый в качестве посредника, для обмена одних благ на другие. При приобретении благ их стоимость вычитается из общего значения денег.

- 7) Специальность – показатель, определяющий то, как будет развиваться линия основного обучения в повествовании.
- 8) Статус военной кафедры – показатель, определяющий то, как будет развиваться линия обучения на военной кафедре в повествовании.

Навыки - исчисляемая характеристика персонажа, показывающая его владение какой-либо областью деятельности или знаний. Навыки отличаются от характеристик тем, что их требуется изучать, в то время как характеристики доступны изначально. Для возможности изучения требуется выполнить необходимые условия, вроде достижения определенного уровня или определенного значения какой-либо одной или нескольких основных характеристик. Чем выше уровень навыка, тем выше требования.

Навыки:

- 1) Дополнительное время – коэффициент, определяющий дополнительное значение времени, отводимое на выполнение игры-головоломки или этапа повествования.
- 2) Пониженная сложность - обозначает вероятность, с которой возможно понизить уровень проходимой игры-головоломки.
- 3) Автопобеда - обозначает вероятность, с которой возможно выиграть игру-головоломку без прохождения.
- 4) Замедленное старение – уменьшает количество здоровья, затрачиваемое на прохождение игры-головоломки, а также уменьшает количество теряемого здоровья, при нахождении в открытом мире.
- 5) Ускоренный перекур – уменьшает количество времени, необходимое для перезарядки проигранной игры-головоломки.
- 6) Любимчик – увеличивает число баллов за успешное прохождение игры-головоломки.

После определения основных объектов можно описать функцию 4. Управление характеристиками персонажа – это набор функций:

- 1) Начисление опыта персонажа;

- 2) Начисление денег
- 3) Вычитание денег;
- 4) Начисление репутации;
- 5) Вычитание репутации;
- 6) Перманентное повышение уровня.
- 7) Добавление основных характеристик.
- 8) Добавление второстепенных характеристик.
- 9) Активация навыков.

Функции 1-6 доступны для других подсистем. Функции 7-9 предназначены для внутреннего использования.

В результате работы подсистемы формируется выходная информация в виде значений основных характеристик и показателей для работы других подсистем.

2.3. Проектирование подсистемы игрового мира

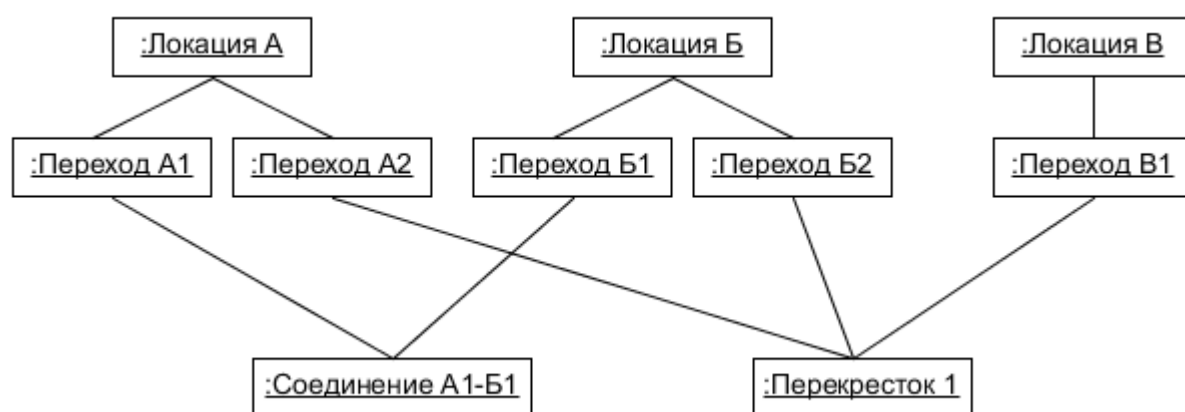
Подсистема игрового мира – обеспечивает существование открытого мира игры, управление персонажем внутри него и взаимодействием с объектами игрового мира.

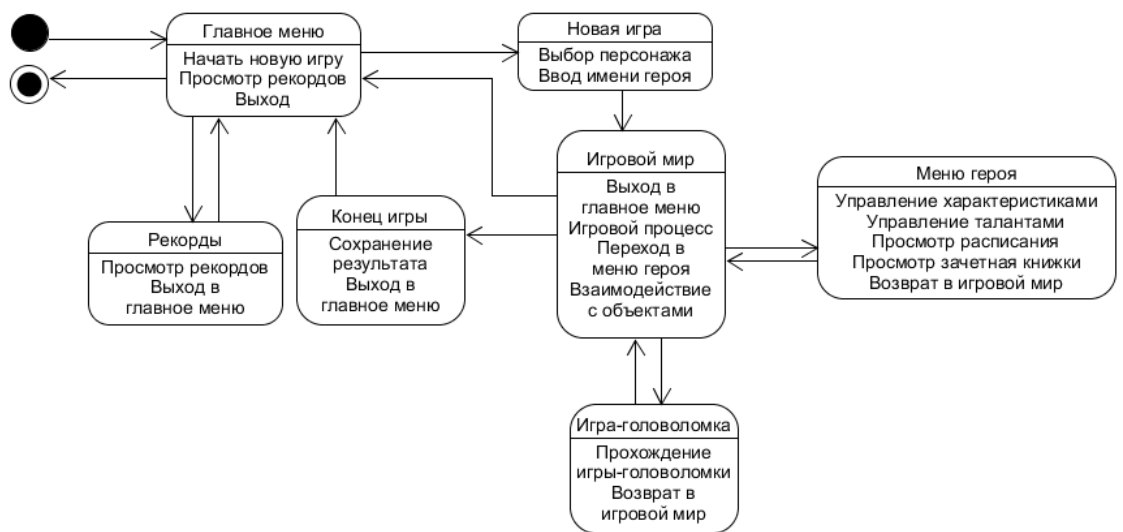
Игровой мир состоит из 5 главных компонентов:

- 1) Локации – области, в которых возможно перемещение персонажа;
- 2) Переходы – объекты, с которыми можно взаимодействовать для путешествия между локациями;
- 3) Соединения – объекты, соединяющие 2 перехода;
- 4) Перекрестки – объекты, соединяющие более 2 переходов;
- 5) Точки входа – объекты, с которыми можно взаимодействовать для выполнения каких-либо задач.

Входными данными для данной подсистемы является информация об игровом мире, представленная в виде данных о каждой группе объектов:

- 1) Для локаций – данные о локациях;
- 2) Для переходов – данные о переходах;





Были выделены следующие игровые состояния:

- 1) Главное меню – первое состояние игровой системы. Оно обеспечивает доступ к началу новой игры, просмотру рекордов и выход.
- 2) Рекорды – позволяет просматривать историю всех прохождений игры. Из этого состояния можно вернуться обратно в главное меню.
- 3) Новая игра – инициализирует начало новой игры. Включает в себя выбор персонажа пользователя и ввод имени персонажа.
- 4) Игровой мир – предоставляет управление передвижением персонажа в игровом мире. Позволяет переходить в меню героя. При взаимодействии с точкой входа в зависимости от результат подсистемы автоматической генерации переходит в игру-головоломку. При прохождении игры осуществляется переход в состояние конца игры. Также из этого состояния можно вернуться в главное меню.
- 5) Меню героя – предоставляет доступ к управлению характеристиками персонажа, талантам, просмотру расписания и завершенных предметов.
- 6) Игра-головоломка – состояние в котором происходит рашение задач-головоломок.
- 7) Конец игры – при переходе в это состояние происходит получение игровой концовки, а также создается запись о прохождении, которую можно просмотреть в рекордах.

2.5. Проектирование подсистемы повествования

Подсистема повествования – обеспечивает сюжетную составляющую игры.

Входными данными для этой подсистемы является выбранная специальность. Всего доступно 9 специальностей:

- 1) Информатика и вычислительная техника;

- 2) Информационные системы и технологии;
- 3) Прикладная информатика;
- 4) Программная инженерия;
- 5) Документоведение и архивоведение;
- 6) Математика;
- 7) Прикладная математика и информатика;
- 8) Прикладная математика;
- 9) Математическое обеспечение и администрирование информационных систем;

Каждой выбранной специальности соответствует свой набор учебных дисциплин (предметов).

Основная сюжетная кампания подразделена на 18 глав (игровых фаз):

- 1) Пролог;
- 2) Учебный семестр (х8, чередуется с сессией);
- 3) Сессия (х8, чередуется учебным семестром);
- 4) Выпускной экзамен.

Во время прохождения этих глав происходит развитие 2 сюжетных линий:

- 1) Линия основного обучения;
- 2) Линия обучения на военной кафедре;

Цель повествования заключается в получении концовки, которая определяется 2 факторами:

- 1) Как была закончена линия основной учебы;
- 2) Как была закончена линия военной кафедры.

При прохождении линии основной учебы возможно получение 1 из 3 финалов:

- 1) Основная учеба завершена на «Удовлетворительно» (ОУ 1);
- 2) Основная учеба завершена на «Хорошо» (ОУ 2);
- 3) Основная учеба завершена на «Отлично» (ОУ 3).

При прохождении линии военной кафедры возможно получении 1 из 5 возможных финалов:

- 1) Военная кафедра завершена на «Удовлетворительно» (ВК 1);
- 2) Военная кафедра завершена на «Хорошо» (ВК 2);
- 3) Военная кафедра завершена на «Отлично» (ВК 3);
- 4) Герой был исключен с военной кафедры (ВК 4);
- 5) Военная кафедра была проигнорирована (ВК 5);

На основе комбинации этих финалов возможно 15 вариантов концовок. Некоторые из них продемонстрированы в пункте 3.7.

Помимо этого, в игре присутствуют «внезапные» концовки:

- 1) Смерть – значение здоровья героя достигает 0;
- 2) Отчисление – герой не успел завершить учебный план семестра, сессии или выпускного экзамена;
- 3) Побег – выбор соответствующей возможности на этапе семестра, сессии или выпускного экзамена.
- 4) Забрать документы - выбор соответствующей возможности на этапе семестра, сессии или выпускного экзамена.
- 5) Уход - выбор соответствующей возможности на этапе пролога.

Каждый этап повествования ограничен по времени – эти таймеры контролируются подсистемой. Существуют следующие таймеры:

- 1) Таймер времени для прохождения этапа. Для разных глав его значение различно: пролог – 10 минут, учебный семестр – 1 час, сессия – 20 минут, выпускной экзамен 15 минут;
- 2) Таймер отсчета стипендии – интервал времени, с которой начисляется денежная стипендия. Выплата стипендии формируется на основе анализа успеваемости предыдущей сессии.
- 3) Таймер отсчета задания для военной кафедры – интервал времени, по которому активируются задания для выполнения линии обучения на военной кафедры.

- 4) Таймер доступа к заданию военной кафедры – интервал времени, в течении которого доступно задание военной кафедры.
- 5) Получение финала игры – если произошла ситуация, приводящая к финалу, подсистема формирует соответствующее сообщение для подсистемы игровых состояний и создает запись в рекордах.

Функции, которые выполняет система:

- 1) Обновление таймеров – управление значениями таймеров текущей игровой фазы и реагирование на них.
- 2) Отслеживание прогресса повествования – отслеживание выполненных/невыполненных заданий текущей фазы, состояние сюжетных линий.
- 3) Установка следующей игровой фазы - когда условия для перехода на следующую фазу выполнены.
- 4) Формирование списка учебных дисциплин текущей специальности для текущей игровой фазы.

Также подсистема осуществляет управление над некоторыми точками входа, влияющими на повествование:

- 1) Деканат;
- 2) Главный выход;
- 3) Расписание.

Выходными данными этой подсистемы являются:

- 1) Список учебных дисциплин, актуальных для текущей игровой фазы, доступный для других подсистем;
- 2) Данные для подсистемы игровых состояний в случае смены фазы или конца игры.

2.6. Проектирование подсистемы игр-головоломок

Подсистема игр-головоломок – обеспечивает управление играми-головоломками.

Всего в игровой системе представлено 5 игр-головоломок:

- 1) Викторина;
- 2) Мозаика;
- 3) Полоса препятствий;
- 4) QTE – испытание;
- 5) Пары.

Входными данными для данной системы является информация, описывающая стартовую позицию для игры-головоломки. В нее входят:

- 1) Тип игры – головоломки;
- 2) Сложность игры-головоломки (требуется для формирования результат при завершении игры-головоломки);
- 3) Набор факторов для генерации стартовой позиции (подробно об этих факторах описано в пункте 2.7)

После генерации стартовой позиции идет непосредственно решение головоломки игроком.

По результатам решения головоломки формируются выходные данные:

- 1) Количество баллов по 12 бальной шкале (предназначены для системы повествования)
- 2) Награды в виде опыта, репутации, денег в случае победы. В зависимости от сложности головоломки награды разные по величине.

Сама подсистема представлена в виде 2 объектов:

- 1) Объект, управляющий работой активной игры-головоломки;
- 2) Объект, реализующий игру головоломку.

Главным преимуществом такого подхода является расширяемость подсистемы за счет реализации новых игр-головоломок на основе базового объекта.

Краткое описание игр-головоломок.

Викторина – тест, состоящий из нескольких вопросов с несколькими вариантами ответа. Условия победы: правильно ответить на большую часть вопросов за отведенное время.

Мозаика – задача по собиранию картинки. Стартовая позиция: изображение разбито на небольшие квадратные фрагменты, перемешанные между собой, а некоторые фрагменты перевернуты или отражены. Условия победы – собрать изображение за отведенное время с помощью операции перестановки фрагментов, поворота фрагмента, отражения фрагмента.

Полоса препятствий – задача по преодолению области, содержащие динамические препятствия. Стартовая позиция: игрок находится в красной зоне, полоса препятствий представлена в виде движущихся прямоугольных объектов, разного размера и с разной скоростью по вертикали, горизонтали и диагонали. Условия победы – достигнуть зеленой зоны, пока у игрока не закончилась жизнь, и не вышло время.

QTE – испытание – задача по угадыванию клавиш. Стартовая позиция: движущиеся элементы, соответствующие клавишам на компьютере, движутся через зеленую зону. Условия победы – набрать необходимое количество очков, нажимая кнопки на клавиатуре, которые соответствуют клавишам, лежащим в зеленой зоне.

Пары – задача по нахождению одинаковых коллекций. Стартовая позиция: большое число элементов, скрытые за «масками», сгруппированные в коллекции, расставленные в случайном порядке. При открытии всех элементов одной коллекции она убирается. Условия победы – убрать все пары, пока не вышло время.

2.7. Проектирование подсистемы автоматической генерации игровых ситуаций

После того, как в предыдущем разделе были спроектированы основные игровые подсистемы, можно приступить к проектированию системы автоматической генерации игровых ситуаций.

Так как основная цель игры – путешествие по локациям и решение игр-головоломок, то и задача проектируемой системы - генерация пар «Точка входа» - «Игра-головоломка».

Входные данные, необходимые подсистеме:

- 1) Список доступных точек входа;
- 2) Список активных учебных дисциплин для генерации расписания.

Саму задачу генерацию пар можно разбить на 2 подзадачи:

- 1) Задача генерации точек входа;
- 2) Задача генерации стартовых позиций игр – головоломок.

Генерация точек входа.

Целью данной задачи является создание функции, которая свяжет все задания текущих учебных дисциплин, которым требуется генерация с некоторыми точками входа.

У задачи есть 1 важное условие: в любой момент игры ни одна точка входа не должна принадлежать более чем 1 паре.

Если представить, что E – подмножество всех точек входа, $E_z \subseteq E$ – подмножество уже занятых в парах точек входа, $E_c \subseteq E$ – подмножество свободных точек входа, то генерация точек входа сводится к выборке из подмножества E_c подмножества $E_b = \{e_1, e_2 \dots e_n\}$, где n – число выбранных элементов. Далее необходимо выбранные точки входы привязать к учебным дисциплинам, а элементы подмножества E_b установить в E_z .

На основании полученных выводов была построена диаграмма активностей (приложение A1).

Описание алгоритма:

Из подсистемы повествования получаем список активных учебных дисциплин. Для каждой дисциплины проверяется статус генерации. Так как на момент выполнения генерации у некоторых предметов уже были завершены активные задания, то необходимо освободить соответствующие точки входа.

Затем, для каждого активного предмета выполняется следующий набор действий:

- 1) Проверяется, нуждается ли предмет в генерации. Если нет, то переходим в пункт 5.
- 2) Проверяется, нуждается ли предмет в полной генерации. Если нет, то переходим в пункт 4.
- 3) Присвоение предмету набора игр-головоломок. С помощью генератора случайных чисел выбираются 3 уникальных числа от 0 до 5 (количество игр-головоломок).
- 4) Каждый тип игры-головоломки устанавливается предмету с определённым долей выпадения при генерации стартовой ситуации. Для первой игры она составляет 50-65%, для второй 25-30, для третьей – все остальное. Данное действие вносит дополнительный элемент случайности в генерацию стартовых позиций. После присвоения игр предмет устанавливается в состояние генерации – «Нужна регенерация»
- 5) Проверяется, нуждается ли предмет в регенерации. Если нет, то переходим к пункту 8.
- 6) Привязка предмета к точке входа. Из подсистемы игрового мира получаем список свободных точек входа.
- 7) Выбираем с помощью генератора случайных чисел одну из точек.
- 8) Устанавливаем эту точку входа в состояние занята.
- 9) Устанавливаем предмету номер новой точки входа.
- 10) Выбираем следующий предмет в списке. Если больше предметов нет – завершаем выполнение алгоритма.

В результате выполнения алгоритма был получен первый элемент пары - «Точка входа».

Генерация точек входа необходима как минимум после выполнения волны заданий активных предметов.

Генерация стартовых позиций игр-головоломок.

Целью данной задачи является создание такой функции генерации $f(x_1, x_2, \dots, x_n)$, где $x_1, \dots, x_n$ – факторы, влияющие на генерацию, которая позволяла бы создавать стартовые позиции с минимальной вероятностью повторения. Данного результата можно достичь 2 способами:

- 1) Использование большого числа факторов генерации;
- 2) Применение сложных алгоритмов для генерации.

Факторы, влияющие на результат генерации можно разделить на 2 группы:

- 1) Статические факторы - факторы, формирование которых происходит вне функции генерации стартовых позиций;
- 2) Динамические факторы – факторы, формирование которых происходит непосредственно перед генерацией стартовой ситуации игры-головоломки, представленные в виде диапазонов значений. У каждой игры-головоломки свой набор таких факторов.

К статическим факторам относятся:

- 1) Сложность – определяет набор динамических факторов, влияющих на генерацию. У всех игр-головоломок, существует 3 вида сложности.
- 2) Навык «Пониженная сложность»;
- 3) Навык «Автопобеда».

Данные статические факторы являются общими для всех игр-головоломок. Также присутствуют статические факторы для конкретных игр-головоломок. А так как у каждой игры-головоломки наборы факторов отличаются, то и функций генерации будет несколько.

Функция генерации для игры-головоломки «Викторина».

Индивидуальные статические факторы, влияющие на генерацию:

- 1) Навык «Дополнительное время».

Динамические факторы, влияющие на генерацию:

- 1) Набор вопросов – общий фактор, независимый от сложности, формируется внутри игры;
- 2) Порядок вопросов – общий фактор, независимый от сложности, формируется внутри игры.

Так как набор и значения динамических факторов зависит от сложности, то для удобства они представлены в таблице 4.

Таблица 4 - Динамические факторы для игры "Викторина".

	Легко	Средне	Сложно
Количество вариантов ответа	От 2 до 4	От 3 до 6	От 4 до 8
Количество вопросов	От 5 до 10	От 7 до 15	От 10 до 20
Время на выполнение	120 секунд	180 секунд	240 секунд

На основе анализа динамических факторов была построена блок-схема (приложение А2).

Функция генерации для игры-головоломки «Мозаика».

Индивидуальные статические факторы, влияющие на генерацию:

- 1) Навык «Дополнительное время».

Динамические факторы, влияющие на генерацию:

- 1) Изображение – фактор, независимый от сложности.

Ниже представлена сравнительная таблица зависимости динамических факторов от сложности (таблица 5).

Таблица 5 - Динамические факторы для игры "Мозаика".

	Легко	Средне	Сложно
Класс изображения	1	2	3
Рядов	6	7	8
Колонок	8	10	12
Размер элемента	100	85	75
Значение отражения для каждого элемента	-	0 или 1	0 или 1
Значение поворота для каждого элемента	-	-	От 0 до 3
Время на выполнение	150 секунд	300 секунд	600 секунд

На основе анализа динамических факторов была построена блок-схема алгоритма генерации стартовых позиций (приложение А3).

Функция генерации для игры-головоломки «Полоса препятствий».

Индивидуальные статические факторы, влияющие на генерацию:

- 1) Бонус времени на решение – множитель времени, отводимого на выполнение задания. Зависит от характеристики «Выносливость».
- 2) Дополнительно число жизней – определяет число дополнительных допустимых смертей, не приводящих к поражению. Зависит от характеристики «Выносливость».

Динамические факторы, влияющие на генерацию:

Ниже представлена таблица зависимости динамических факторов от сложности (таблица 6).

Таблица 6 - Динамические факторы для игры "Полоса препятствий".

	Легко	Средне	Сложно
Количество жизней	5	4	3
Направление движения препятствий	Вертикальное	Вертикальное, горизонтальное	Вертикальное, горизонтальное, диагональное
Количество препятствий	От 6 до 10	От 6 до 10; От 5 до 8	От 6 до 10; От 5 до 8; От 3 до 6
Начальное положение препятствий	От 40 до 520	От 40 до 520 От 80 до 880	От 40 до 520 От 80 до 880
Скорость движения препятствий	От 1 до 13	От 1 до 13 От 1 до 10	От 1 до 13 От 1 до 10 От 1 до 7
Размер препятствий	От 50 до 200	От 50 до 200; От 50 до 400	От 50 до 200; От 50 до 400; От 50 до 300
Время на выполнение	30 секунд	45 секунд	60 секунд

На основе анализа динамических факторов была построена блок-схема алгоритма генерации стартовых позиций (приложение А4).

Функция генерации для игры-головоломки «QTE-испытание».

Индивидуальные статические факторы, влияющие на генерацию:

- 1) Проходные пороги – количества баллов, которое необходимо набрать для выполнения задания. Зависит от характеристики «Интеллект».

Динамические факторы, влияющие на генерацию:

Ниже представлены таблица зависимости динамических факторов от сложности (таблица 7).

Таблица 7 - Динамические факторы для игры "Qte-испытание".

	Легко	Средне	Сложно
Число элементов	50	100	150
Количество рядов	1	2	3
Количество клавиш	4	8	12
Скорость движения элементов	От 9 до 12	От 7 до 10	От 5 до 8
Минимальное число баллов для прохождения	25	50	75
Начальное положение ворот	605	От 220 до 1080	От 220 до 1080
Скорость движения ворот	-	-	От 1 до 5

На основе анализа динамических факторов была построена блок-схема алгоритма генерации стартовых позиций (приложение А5).

Функция генерации для игры-головоломки «Пары».

Индивидуальные статические факторы, влияющие на генерацию:

- 1) Бонус времени на решение – множитель времени, отводимого на выполнение задания. Зависит от характеристики «Выносливость».
- 2) Дополнительные слоты предпросмотра – увеличивает число элементов, которые можно открыть до сброса коллекции. Зависит от характеристики «Интеллект».

Динамические факторы, влияющие на генерацию:

Ниже представлены таблицы зависимости динамических факторов от сложности (таблица 8).

Таблица 8 - Динамические факторы для игры "Пары".

	Легко	Средне	Сложно
Элементов в коллекции	От 2 до 4	От 3 до 6	От 4 до 8
Число рядов	5	7	9
Число колонок	10	10	10
Слотов предпросмотра	4	6	8
Время на выполнение	120 секунд	240 секунд	480 секунд

На основе анализа динамических факторов была построена блок-схема алгоритма генерации стартовых позиций (приложение А6).

2.8. Выводы по разделу

В результате проектирования были достигнуты следующие цели:

- 1) Определены функции и действующие лица игровой системы;
- 2) Определены основные компоненты системы;
- 3) Определены входные и выходные данные каждой подсистемы;

При выполнении проектирования игровой системы получены следующие результаты:

- 1) Диаграмма прецедентов игровой системы;
- 2) Диаграмма взаимодействия компонентов системы;
- 3) Диаграмма взаимодействия объектов игрового мира;
- 4) Диаграмма игровых состояний игровой системы;
- 5) Блок-схема алгоритма генерации точек входа;
- 6) Блок-схема алгоритма генерации стартовых позиций для игры-головоломки «Викторина»;
- 7) Блок-схема алгоритма генерации стартовых позиций для игры-головоломки «Мозаика»;
- 8) Блок-схема алгоритма генерации стартовых позиций для игры-головоломки «Полоса препятствий»;
- 9) Блок-схема алгоритма генерации стартовых позиций для игры-головоломки «Qte-испытание»;
- 10) Блок-схема алгоритма генерации стартовых позиций для игры-головоломки «Пары».

После проектирования можно приступать к программной реализации игровой системы.

3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРОВОЙ СИСТЕМЫ

Программная реализации – процесс кодирования алгоритмов на языке программирования. В данном случае используется объектно-ориентированный язык программирования C#.

В данном разделе описываются особенности реализации игровой системы, а также демонстрируется контрольный пример работы всей игровой системы и подсистемы генерации игровых ситуаций, в частности.

3.1. Программная реализация подсистемы развития героя

Данная подсистема реализована в виде класса CharacterAttributes. Ниже представлена диаграмма класса (рисунок 5).

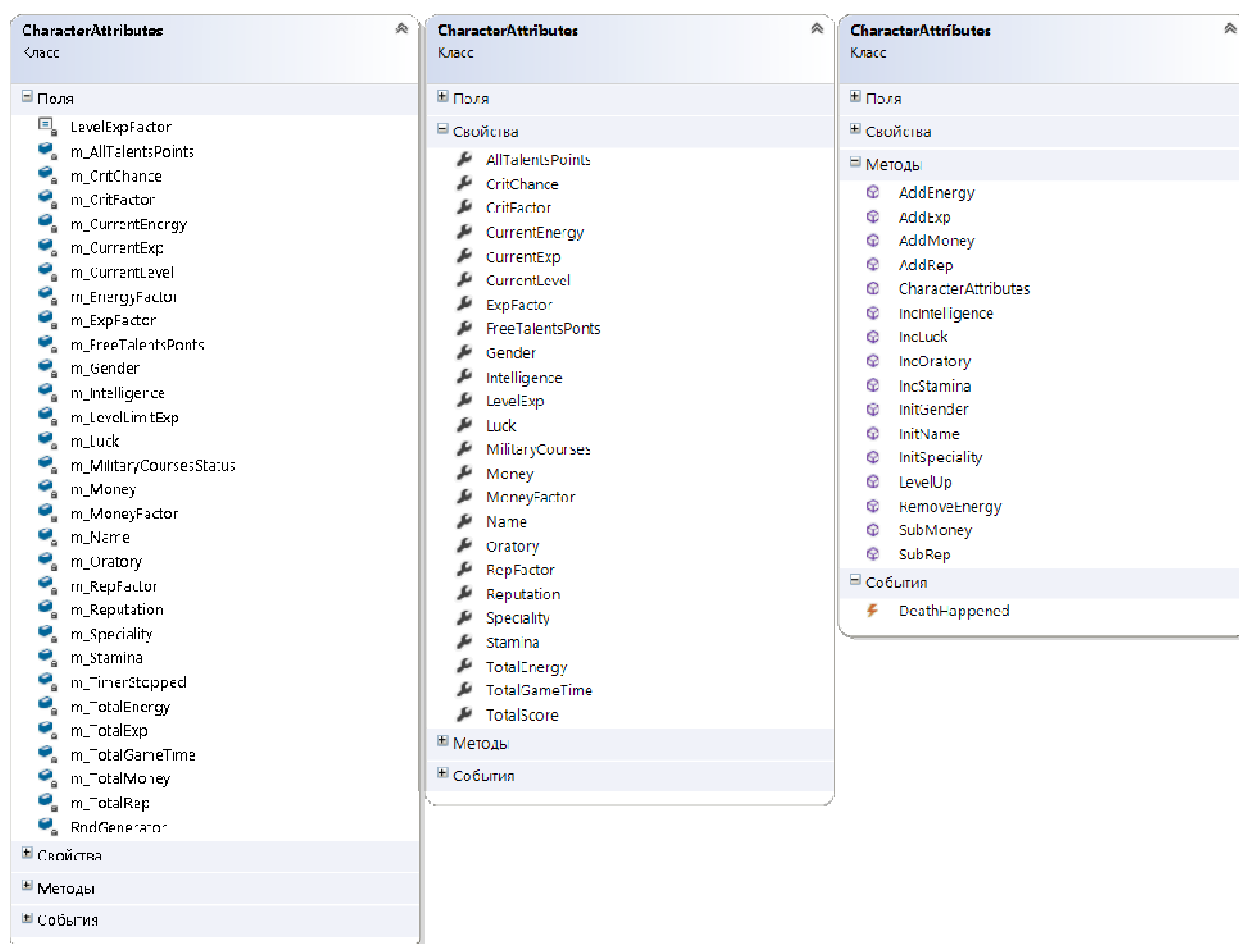


Рисунок 5 - Диаграмма классов подсистемы развития героя.

В полях класса реализованы все ранее спроектированные компоненты: основные характеристики, второстепенные характеристики, показатели.

С помощью свойств классы других подсистем могут получать информацию о всех характеристиках персонажа.

Описание методов:

- 1) AddEnergy – метод, позволяющий увеличивать текущий запас здоровья вплоть до максимального значения;
- 2) RemoveEnergy – метод, позволяющий уменьшать запас текущий запас здоровья. Если значение станет равно 0 – будет вызвано событие DeathHappened, которое сообщит подсистеме повествования что игра закончена.
- 3) AddMoney/SubMoney – методы, позволяющие добавлять/уменьшать количество денег персонажа на какое либо значение.
- 4) AddRep/SubRep – методы, позволяющие добавлять/уменьшать репутацию персонажа на какое либо значение.
- 5) IncIntellegence/IncStamina/IncOratory/IncLuck – увеличивающие Интеллект/Выносливость/Красноречие/Удачу на 1 очко. Также внутри метода изменяются значения и второстепенных характеристик.
- 6) InitName/InitGender/IniSpeciality – методы для инициализации имени персонажа, пола, специальности.
- 7) LevelUp – метод, который увеличивает уровень персонажа на 1.

3.2. Программная реализация подсистемы игрового мира

Подсистема игрового мира представлена следующими классами:

- 1) OpenWorldState – класс, в котором происходит управление компонентами игрового мира;
- 2) Location – класс, описывающий локацию;
- 3) Connection – класс описывающий объект, соединяющий 2 локации;
- 4) Crossroad – класс, описывающий объект, соединяющий более 2 локаций;

5) Transition – класс, описывающий связь «Локация» - «Переход» и «Локация» - «Перекресток».

6) EntryPoint – класс, описывающий точку входа;

7) CharacterPosition – класс для управления позицией персонажа в открытом мире.

Данные о локациях, соединениях, перекрестках, переходах и точках входа содержатся в файлах специального формата.

Диаграмма классов, входящих в состав подсистемы представлена на рисунке 6.

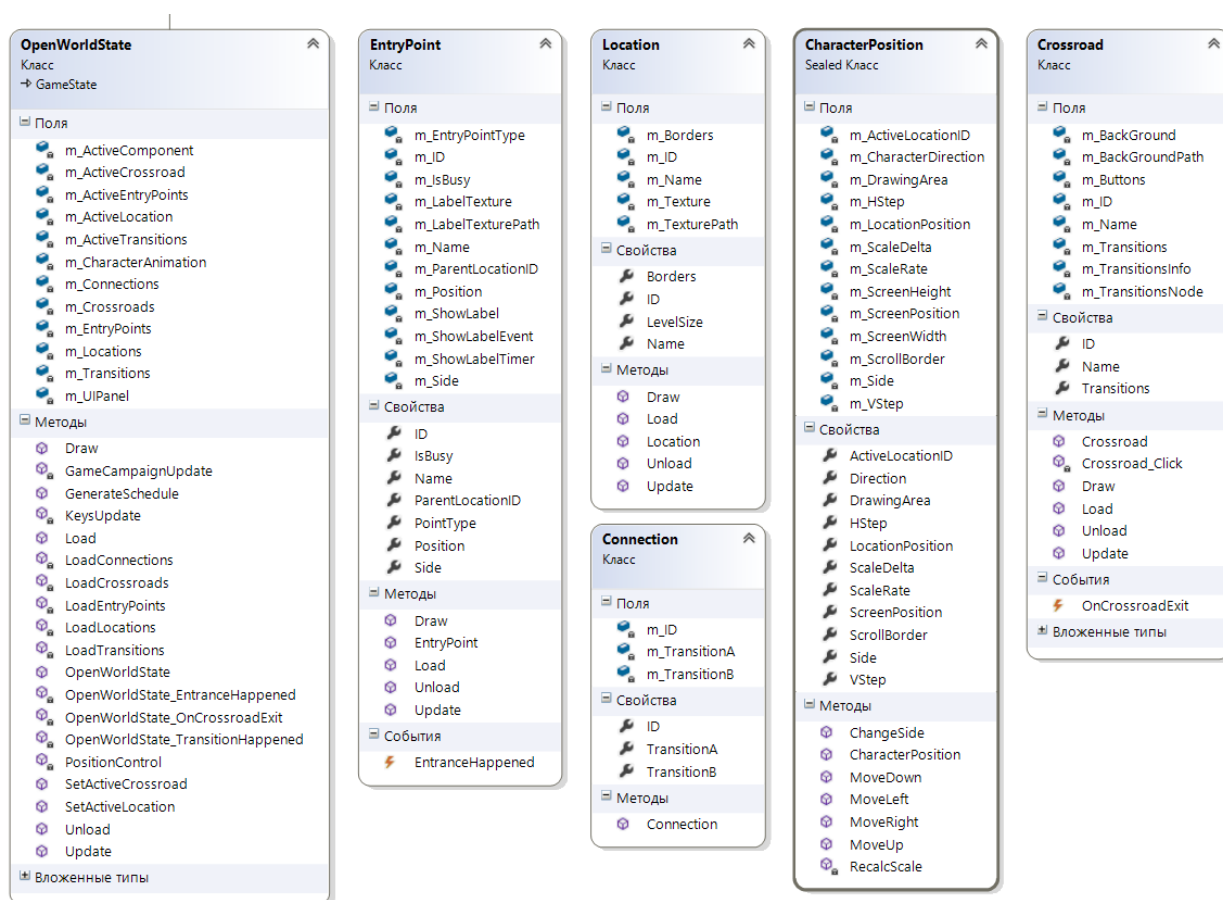


Рисунок 6 - Диаграмма классов подсистемы игрового мира.

Принцип работы:

Класс OpenWorldState является производным от класса GameState.

Когда у класса OpenWorldState вызывается метод Update, то:

1) Обрабатывается клавиши передвижения;

- 2) Анализируется положение персонажа в текущей локации с целью определения взаимодействия с точками входа или переходами;
- 3) Обрабатываются таймеры подсистемы повествования;

Когда вызывается метод Draw, то:

- 1) Происходит прорисовка заднего плана – локации;
- 2) Прорисовывается персонаж – в зависимости от нажатых клавиш передвижения он может находиться в состоянии покоя или в движении;
- 3) Прорисовывается панель героя;
- 4) Отображаются сообщения о возможности взаимодействия с точками входа или переходами.

Преимущества данного решения:

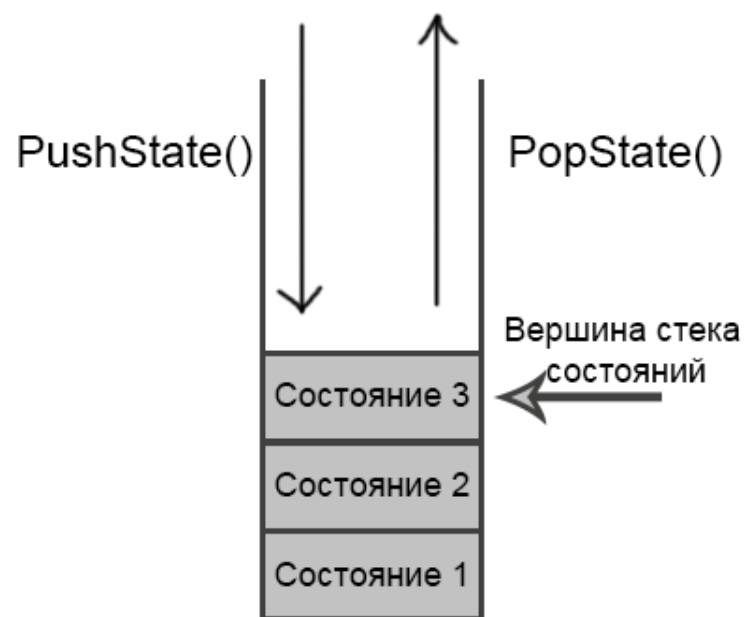
- 1) Вся информация о игровом мире хранится во внешних файлах специального формата;
- 2) Возможность редактирования файлов изменения границ, точек взаимодействия;
- 3) Загрузка спрайтов локаций только в тот момент, когда активна эта локация – уменьшение затрат ресурсов системы;
- 4) Повышение общего быстродействия;
- 5) Удобство расширяемости системы;

3.3. Программная реализация подсистемы игровых состояний

Данная подсистема представлена в виде класса GameStateManager – класс управления состояниями и классом GameState – абстрактный класс состояния игровой системы.

У абстрактного класса GameState 4 метода, которые реализуются в производных классах:

- 1) Load – загрузка ресурсов, необходимых текущему состоянию, инициализация объектов и прочее;
- 2) Update – обработка логики текущего состояния;



3) Повышение общего быстродействия;

4) Удобство расширяемости системы;

Для данной игровой системы были реализованы 25 классов состояний.

Диаграмма этих классов представлена на рисунке ниже (рисунок 8).

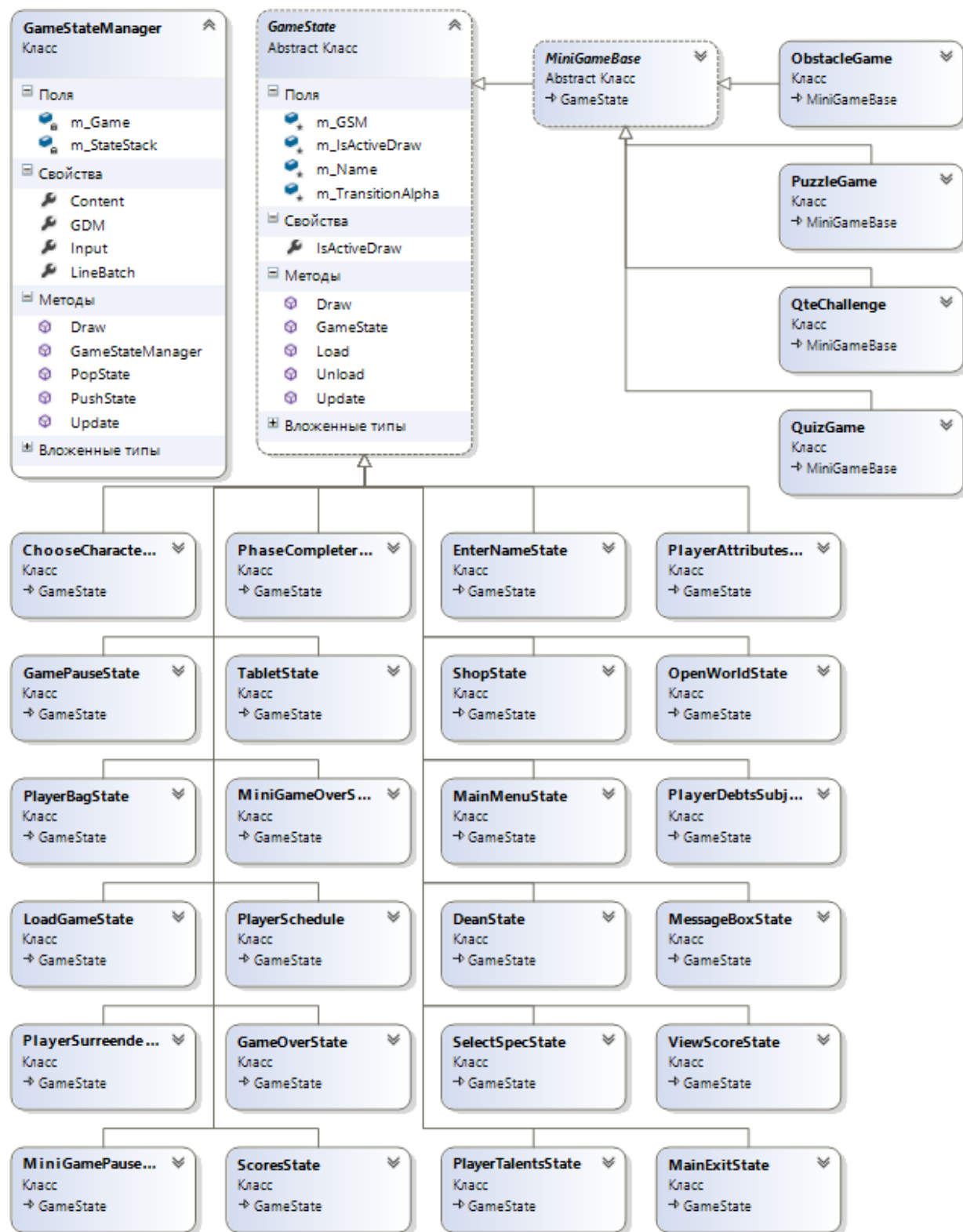


Рисунок 8 - Диаграмма классов подсистемы игровых состояний.

DeanState
Класс
→ GameState

Поля

- m_BackButton
- m_BackGround
- m_SelectSpecialityButton
- m_ShowSelectSpecialityButton
- m_ShowStartDiplomExamButton
- m_ShowStartNextSemesterButton
- m_ShowStartNextSessionButton
- m_ShowTakeDocsAndGoAwayButton
- m_StartDiplomExamButton
- m_StartNextSemesterButton
- m_StartNextSessionButton
- m_TakeDocsAndGoAwayButton

Методы

- DeanState
- Draw
- Load
- m_BackButton_Click
- m_DiplomButton_Click
- m_DocButton_Click
- m_SemesterButton_Click
- m_SessionButton_Click
- m_SpecButton_Click
- Unload
- Update

MainExitState
Класс
→ GameState

Поля

- m_BackButton
- m_BackGround
- m_GoAwayButton
- m_MilitaryQuestButton
- m_MilitaryTimeButton
- m_RunAwayButton
- m_ShowGoAwayButton
- m_ShowMilitaryQuestButton
- m_ShowMilitaryTimeButton
- m_ShowRunAwayButton

Методы

- Draw
- Load
- m_BackButton_Click
- m_GoAwayButton_Click
- m_MilitaryQuestButton_Click
- m_MilitaryTimeButton_Click
- m_RunAwayButton_Click
- MainExitState
- Unload
- Update

Subject
Класс

Поля

- m_AuditoriumName
- m_Availible
- m_CoolDownTimer
- m_ExamValue
- m_GenerationStatus
- m_ID
- m_MiniGamesTable
- m_Name
- m_ParentSpecs
- m_Passed
- m_QuestsCompleted
- m_Score
- PassingScore
- QuestCoolDown
- QuestCount
- s_GlobalRate

Свойства

- AuditoriumName
- Availible
- CoolDownTime
- GenStatus
- Id
- Name

Методы

- ActiveMiniGame_GameOver
- AddPoints
- ContainSemestr
- ContainSpeciality
- SetAuditorium
- SetExamResult
- SetMiniGames
- StartMiniGame
- Subject
- UpdateTimers

Вложенные типы

StoryMode
Класс

Поля

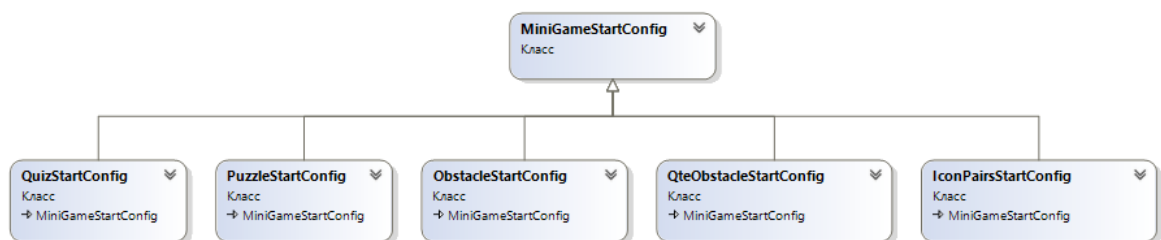
- FinalExamTime
- GrandTimer
- m_ActivePhase
- m_GoAway
- m_Grand
- m_GrandSize
- m_MilitaryCoursesEnabled
- m_MilitaryIntroEnabled
- m_MilitaryTimer
- m_NewPhaseStartedEvent
- m_RemainigTime
- m_RemainingTimeFactor
- m_RemainingTimeOutEvent
- m_RunAway
- m_SelectSpeciality
- m_StartFinalExam
- m_StartNextSemester
- m_StartNextSession
- m_TakeDocsAndGoAway
- MilitaryIntroTimer
- MilitaryResetTimer
- MilitaryTimeInterval
- PrologTime
- SemestrTime
- SessionTime

Свойства

- GoAway
- MilitaryCoursesEnabled
- MilitaryIntroEnabled
- RemainingTime
- RemainingTimeOutEvent
- RunAway
- SelectSpeciality
- StartFinalExam
- StartNextSemester
- StartNextSession
- TakeDocsAndGoAway

Методы

- StartNextPhase
- StoryMode
- UpdateTimers



- 3) MiniGameManager – класс, предназначенный для управления играми-головоломками, реализован по шаблону «Одиночка», поэтому к нему можно получить доступ из любой точки программы.
- 4) MiniGameStartConfig – абстрактный класс, содержащий информацию о стартовой ситуации для игр-головоломок.
- 5) QuizStartConfig, PuzzleStartConfig, ObstacleStartConfig, QteChallengeStartConfig, IconPairsStartConfig – классы, содержащие информацию о стартовой ситуации для конкретных игр-головоломок.

Принцип работы:

- 1) Подсистемой автоматической генерации игровых ситуаций формируется производный объект от MiniGameStartConfig какой-либо игры-головоломки.
- 2) Данный объект передается как аргумент в метод StartMiniGame класса MiniGameManager.
- 3) С помощью ключевого слова «is» в языке проверяется производный класс и затем вызывается соответствующая мини-игра с объектом настройкой в качестве аргумента.

Преимущества данного решения:

- 1) Экономия ресурсов компьютера – при старте мини игры необходимые данные загружаются, а при завершении выгружаются;
- 2) Это повышает общее быстродействие системы;
- 3) Удобство расширяемости;

SoAGoGS

Класс

Поля

-  m_Instance
-  m_RandomGenerator

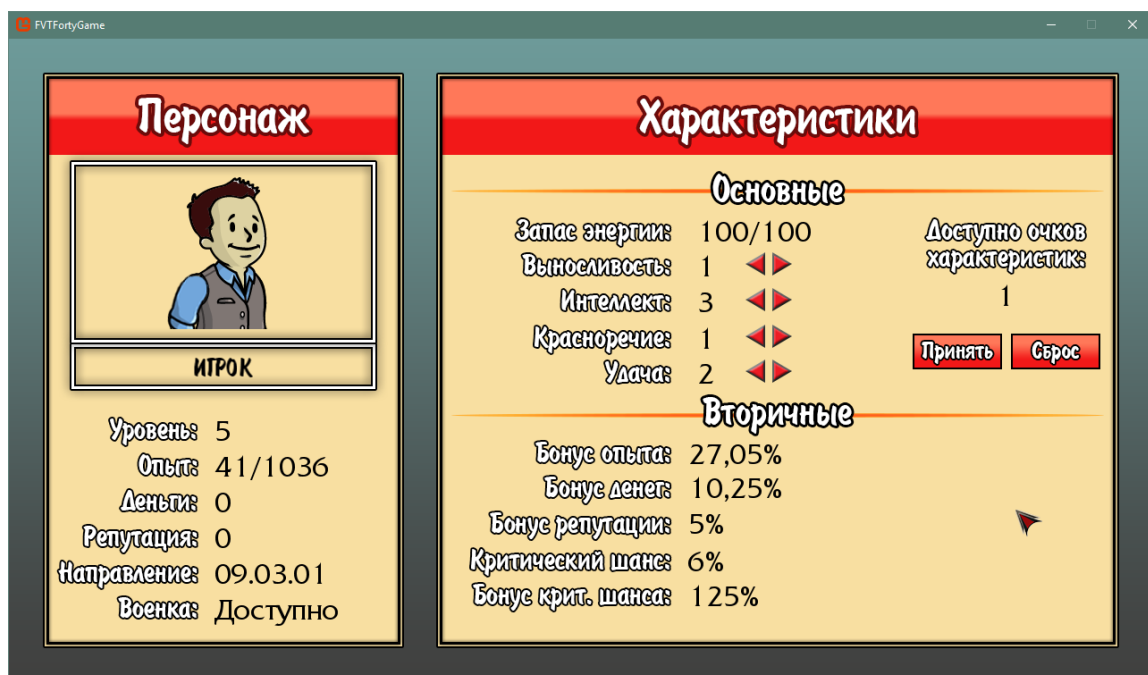
Свойства

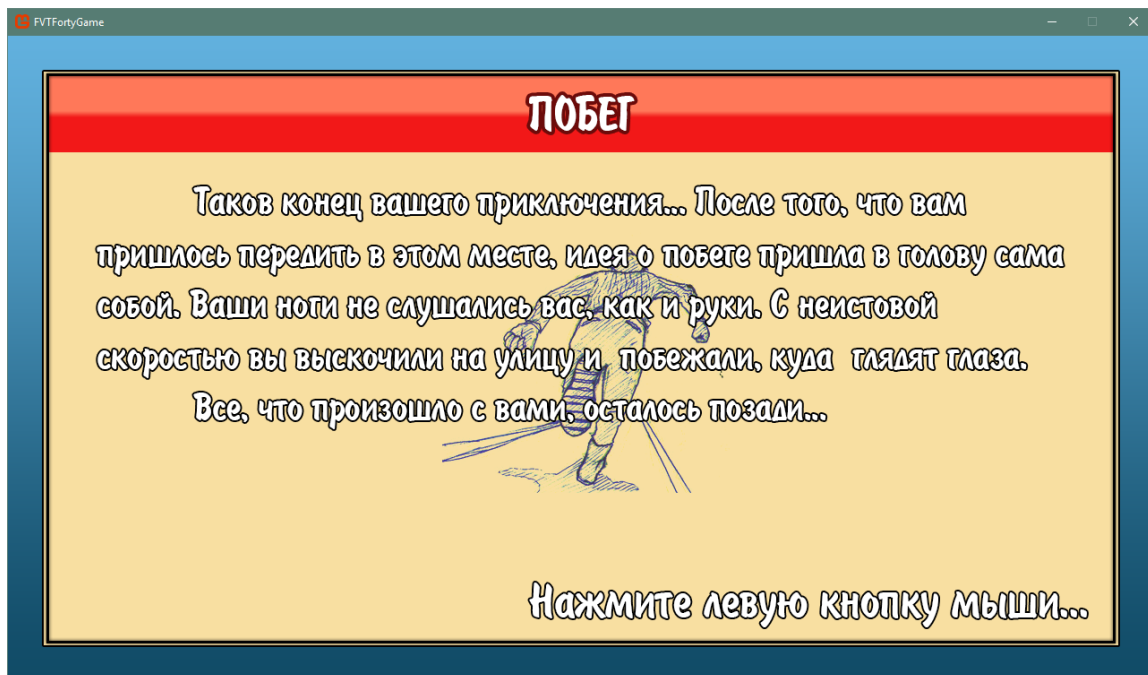
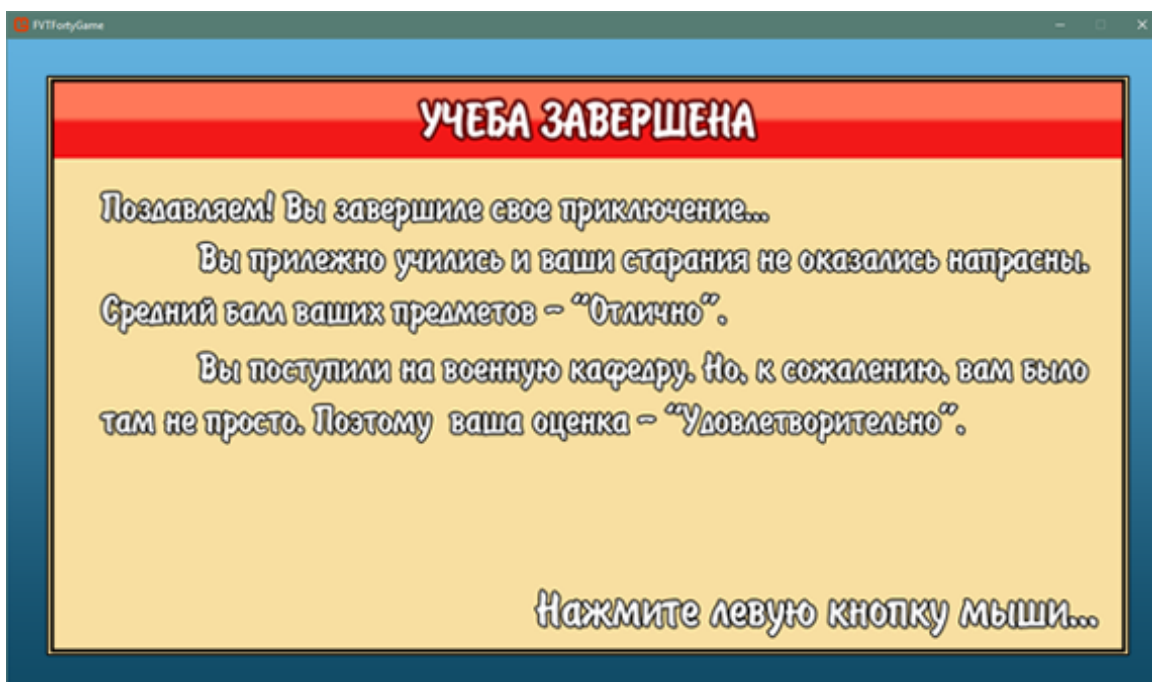
-  Instance

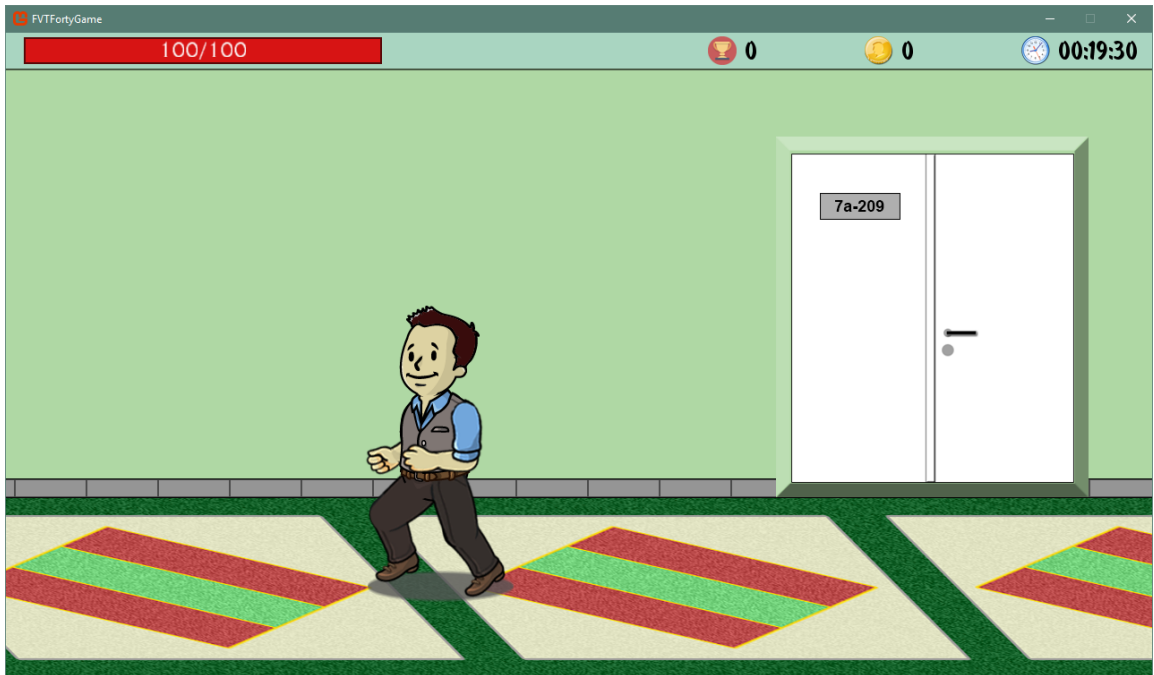
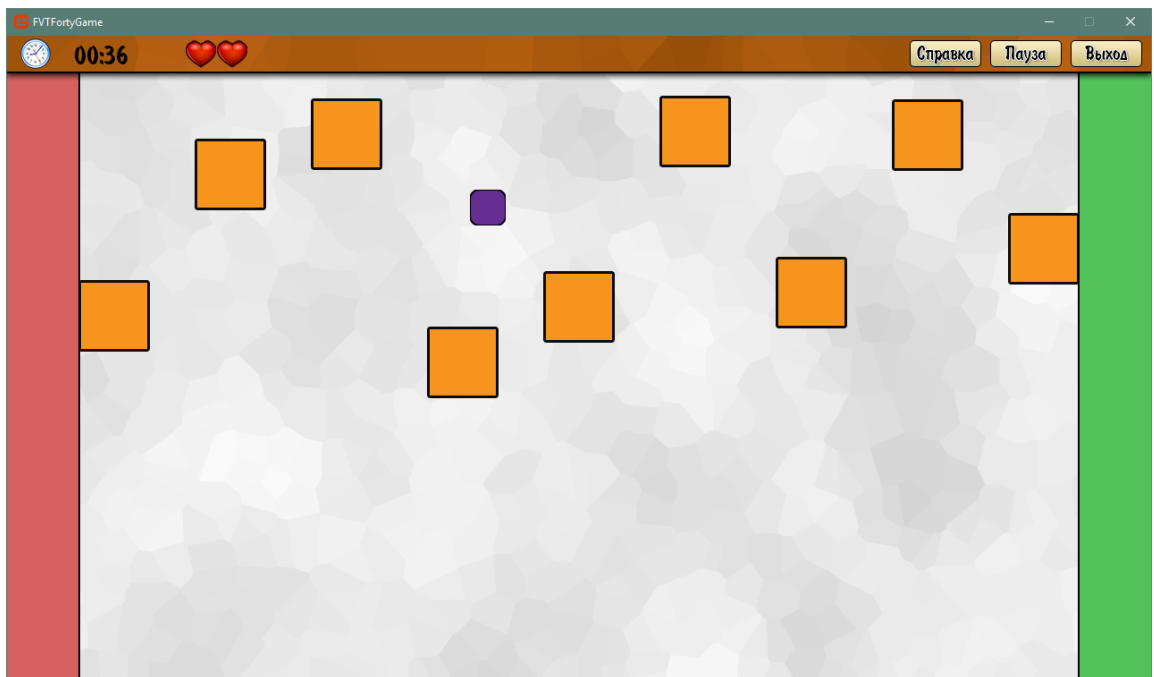
Методы

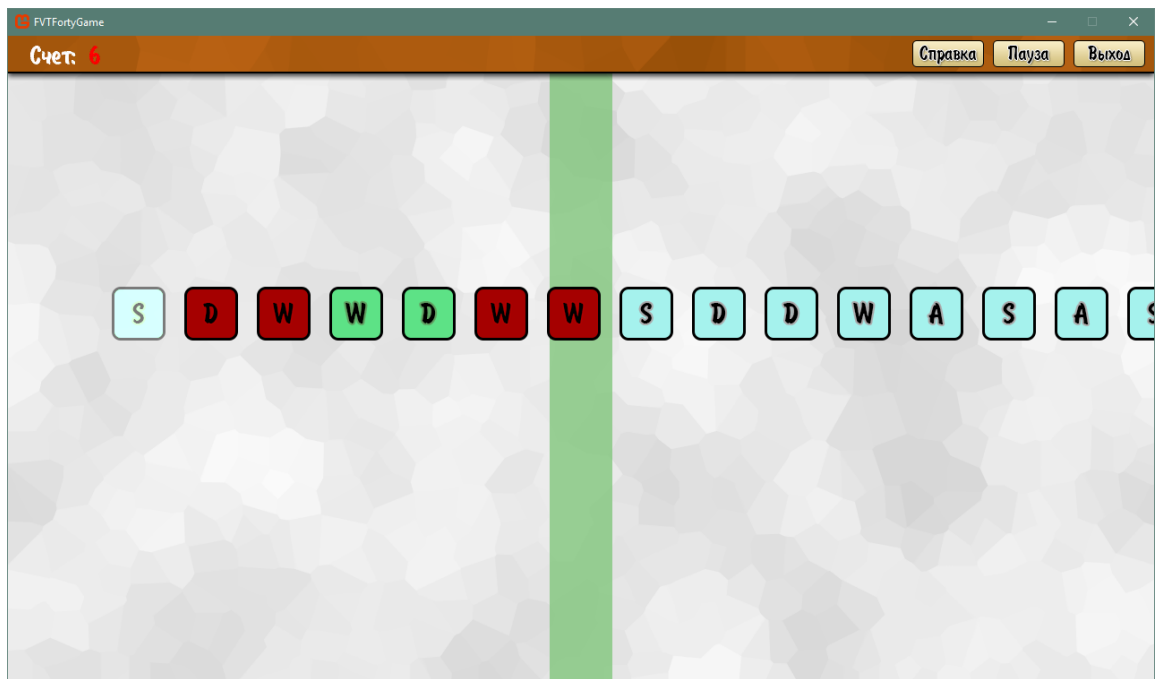
-  GenerateDiagonalObstacle
-  GenerateEntryPoints
-  GenerateHorizontalObstacle
-  GenerateIconPairsStart
-  GenerateObstacleStart
-  GeneratePuzzleStart
-  GenerateQteObstacleStart
-  GenerateQuizStart
-  GenerateVerticleObstacle
-  SoAGoGS

- 2) `GenerateQuizStart` – метод генерации стартовой позиции для игры-головоломки «Викторина». Входными данными для этого алгоритма являются информация о персонаже, необходимая для определения статических факторов и требуемая сложность. Исходный код метода находится в приложении Б2.
- 3) `GeneratePuzzleStart` – метод генерации стартовой позиции для игры-головоломки «Мозаика». Входными данными для этого алгоритма являются информация о персонаже, необходимая для определения статических факторов и требуемая сложность. Исходный код метода находится в приложении Б3.
- 4) `GenerateObstacleStart` – метод генерации стартовой позиции для игры-головоломки «Полоса препятствий». Входными данными для этого алгоритма являются информация о персонаже, необходимая для определения статических факторов и требуемая сложность. Закрытые методы `GenerateHObs`, `GenerateVObs`, `GenerateDObs` предназначены для генерации объектов-препятствий. Исходный код метода находится в приложении Б4.
- 5) `GenerateQteObstacleStart` – метод генерации стартовой позиции для игры-головоломки «Qte-испытание». Входными данными для этого алгоритма являются информация о персонаже, необходимая для определения статических факторов и требуемая сложность. Исходный код метода находится в приложении Б5.
- 6) `GenerateIconPairsStart` – метод генерации стартовой позиции для игры-головоломки «Пары». Входными данными для этого алгоритма являются информация о персонаже, необходимая для определения статических факторов и требуемая сложность. Исходный код метода находится в приложении Б6.









Расписание занятий	
Название предмета	Номер аудитории
Информатика	7Б-204
ИТВИД	7Б-201
Программирование	7Б-202
Иностранный язык	7Б-207
Математика	7Б-203

Пример результата генерации стартовых позиций можно увидеть на рисунке 16 и рисунке 17.

3.8. Выводы по разделу

При выполнении данного раздела были описаны программные реализации, всех спроектированных подсистем. В результате были получены:

- 1) Диаграммы классов, описывающие каждую из подсистем;
- 2) Были сформулированы преимущества реализации для подсистемы игровых состояний, подсистемы игрового мира, подсистемы игрового головоломки.
- 3) Был представлен контрольный пример, в котором наглядно продемонстрирована работоспособность всех игровых подсистем.

ЗАКЛЮЧЕНИЕ

Целью выпускной квалификационной работы являлась разработка системы автоматической генерации игровых ситуаций в рамках RPG-игры. Разработанное программное обеспечение представлено в виде игры «40-летие ФВТ»

Выполнение работы было разделено на 3 раздела:

1) Анализ предметной области.

В ходе выполнения этого раздела были определены системы-конкуренты. Результат сравнения систем был продемонстрирован в виде таблицы (Таблица 1). На основе сравнения была четко определена проблема для ее дальнейшего решения. Далее был проведен анализ языков программирования. По его результатам было решено использовать язык программирования C#, а для, того чтобы упрощения разработки базовых компонентов был использован фреймворк MonoGame.

2) Проектирование игровой системы.

В ходе выполнения данного раздела были определены все основные компоненты подсистемы и выполняемые ими функции. Результатом их проектирования стал набор диаграмм (диаграмма прецедентов, структурная диаграмма взаимодействия, диаграмма игровых состояний, диаграмма взаимодействия объектов игрового мира) и блок схем (блок-схема алгоритма генерации точек входа и блок-схемы алгоритмов генерации стартовых позиций игр-головоломок).

3) Программная реализация игровой системы.

В ходе выполнения раздела были реализованы на языке C# и описаны все подсистемы. В результате были представлены диаграммы классов, описывающие подсистемы. Были раскрыты преимущества реализации для подсистемы игровых состояний, подсистемы игрового мира, подсистемы игр-головоломок. Для подсистемы генерации игровых ситуаций также были представлен листинг программы (приложение Б). Для подтверждения работоспособности игровой системы был продемонстрирован контрольный

пример. Разработанная игровая система обладает большой гибкостью и способна к дальнейшему расширению.

В ходе выполнения выпускной квалификационной работы были продемонстрированы навыки, полученные в ходе обучения в Пензенском государственном университете: умение анализировать предметную область с целью выявления проблем для их решения, умение проектировать ПО и пользоваться средствами объектно-ориентированного моделирования в частности унифицированным языком моделирования (UML), умение программировать на объектно-ориентированном языке программирования (в частности C#), умение писать документацию, соответствующую всем необходимым стандартам.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

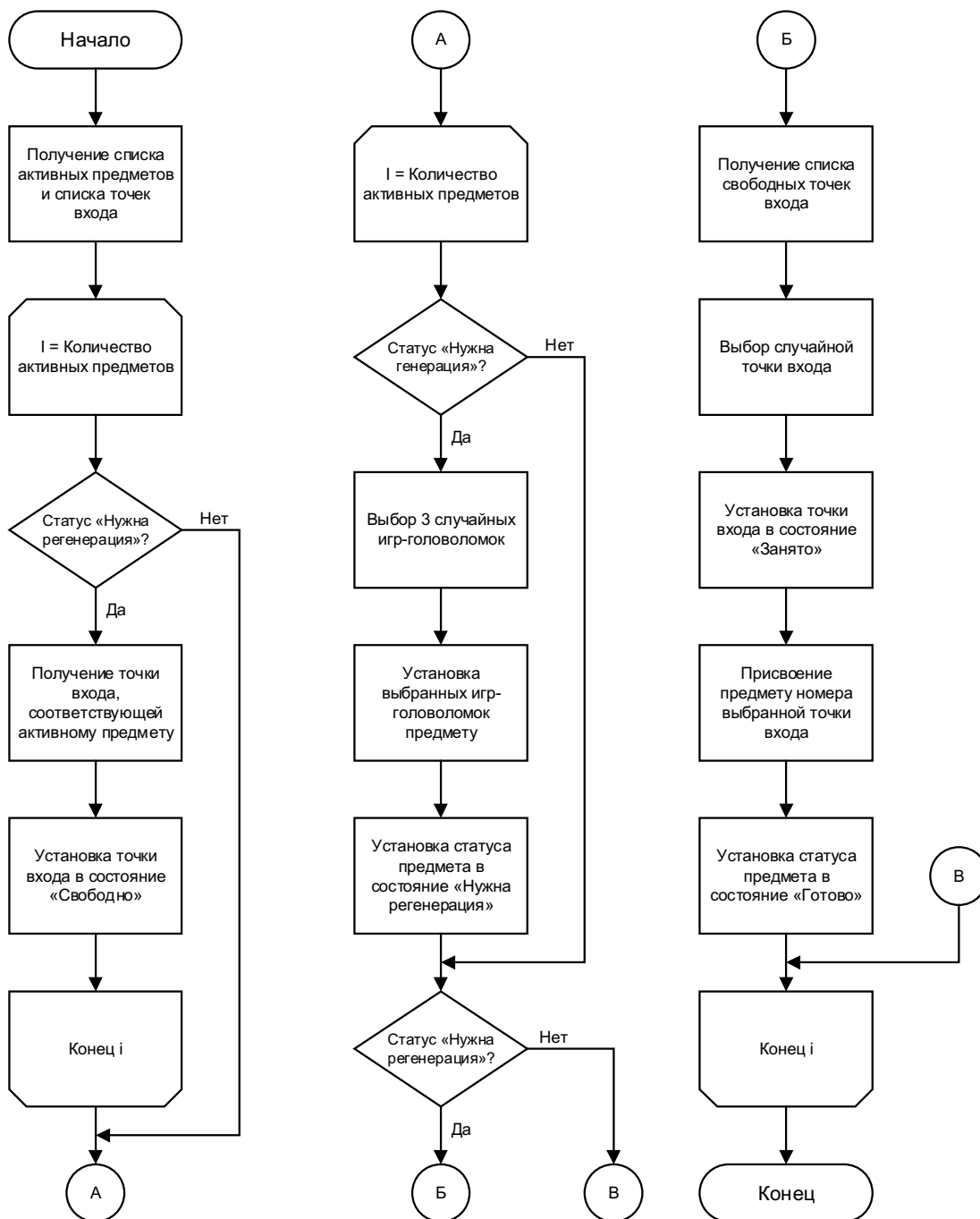
- 1) Компьютерная игра [Электронный ресурс] – URL: https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0 (Дата обращения 26.05.2017)
- 2) Mark Jonson, Generation Next [Электронный ресурс] - URL: <https://www.rockpapershotgun.com/2016/07/22/future-of-procedural-generation-1/> (Дата обращения 26.05.2017)
- 3) About Mark Jonson [Электронный ресурс] - URL: <http://www.ultimaratioregum.co.uk/about-me/> (Дата обращения 26.05.2017)
- 4) Официальный сайт игры Diablo 3 [Электронный ресурс] – URL: <https://eu.battle.net/d3/ru/> (Дата обращения 26.05.2017)
- 5) Официальный сайт игры The Elder Scroll V: Skyrim [Электронный ресурс] – URL: <https://elderscrolls.bethesda.net/> (Дата обращения 26.05.2017)
- 6) Официальный сайт игры The Witness [Электронный ресурс] – URL: <http://the-witness.net/> (Дата обращения 26.05.2017)
- 7) C++ [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/C%2B%2B> (Дата обращения 26.05.2017)
- 8) C# [Электронный ресурс] – URL: https://ru.wikipedia.org/wiki/C_Sharp (Дата обращения 26.05.2017)
- 9) Java [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Java> (Дата обращения 26.05.2017)
- 10) Троелсен Э. "Язык программирования C# 5.0 и платформа .NET 4.5" Вильямс, 2013 год, 1311 с.
- 11) Рихтер Дж. «CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#». — 4-е изд. — СПб.: Питер, 2013. — 896 с.
- 12) Стилмен Э. «Head First. Изучаем C#» / Стилмен Э., Грин Д. — 3-е изд. — СПб.: Питер, 2017.

- 13) Microsoft Visual Studio [Электронный ресурс] – URL: https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio (Дата обращения 26.05.2017)
- 14) MonoGame Documentation [Электронный ресурс] – URL: <http://www.monogame.net/documentation/?page=main> (Дата обращения 26.05.2017)
- 15) Проектирование [Электронный ресурс] – URL: https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5#cite_note-s5-2 (Дата обращения 1.06.2017)
- 16) UML [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/UML> (Дата обращения 26.05.2017)

ПРИЛОЖЕНИЯ

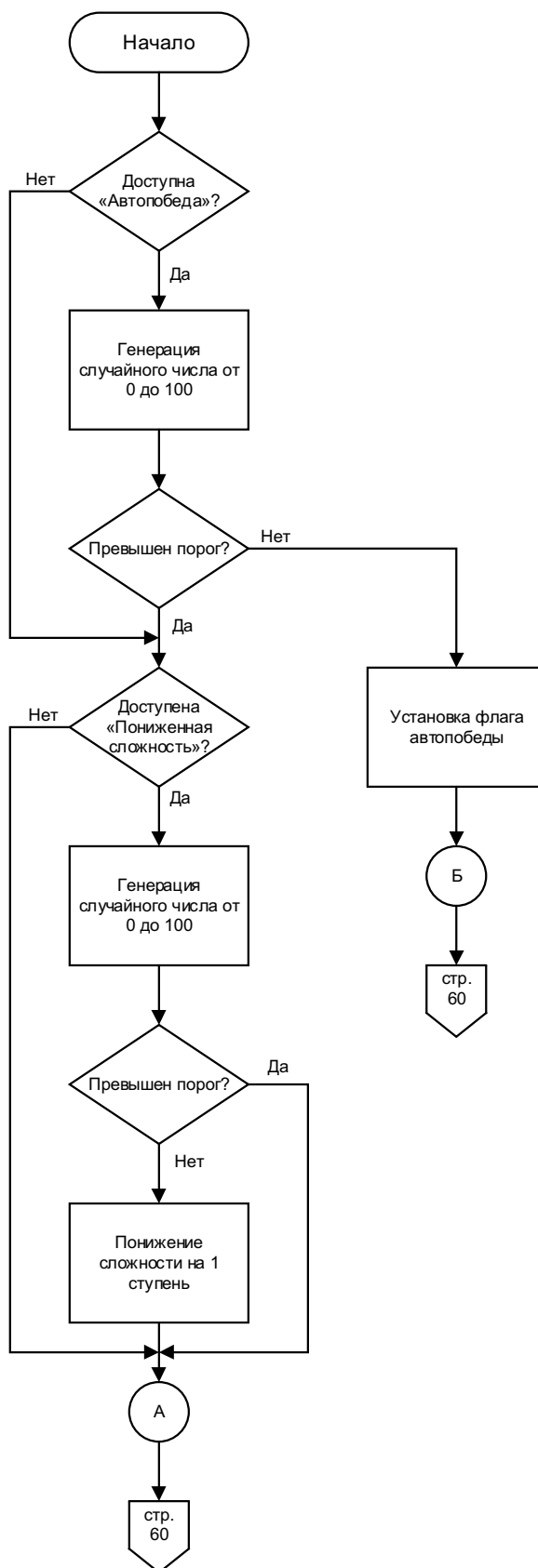
ПРИЛОЖЕНИЕ А1

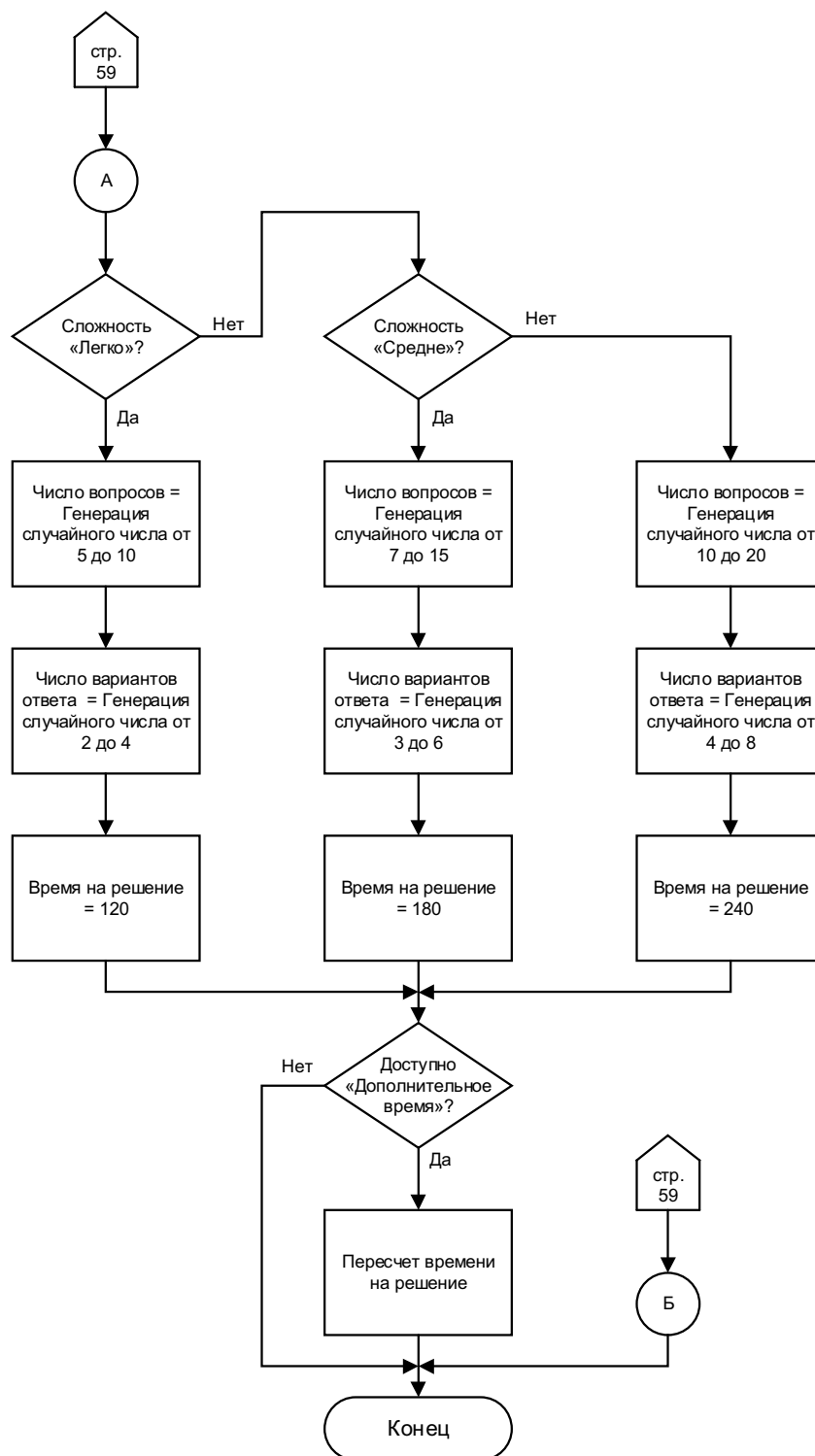
Блок-схема алгоритма генерации точек входа



ПРИЛОЖЕНИЕ А2

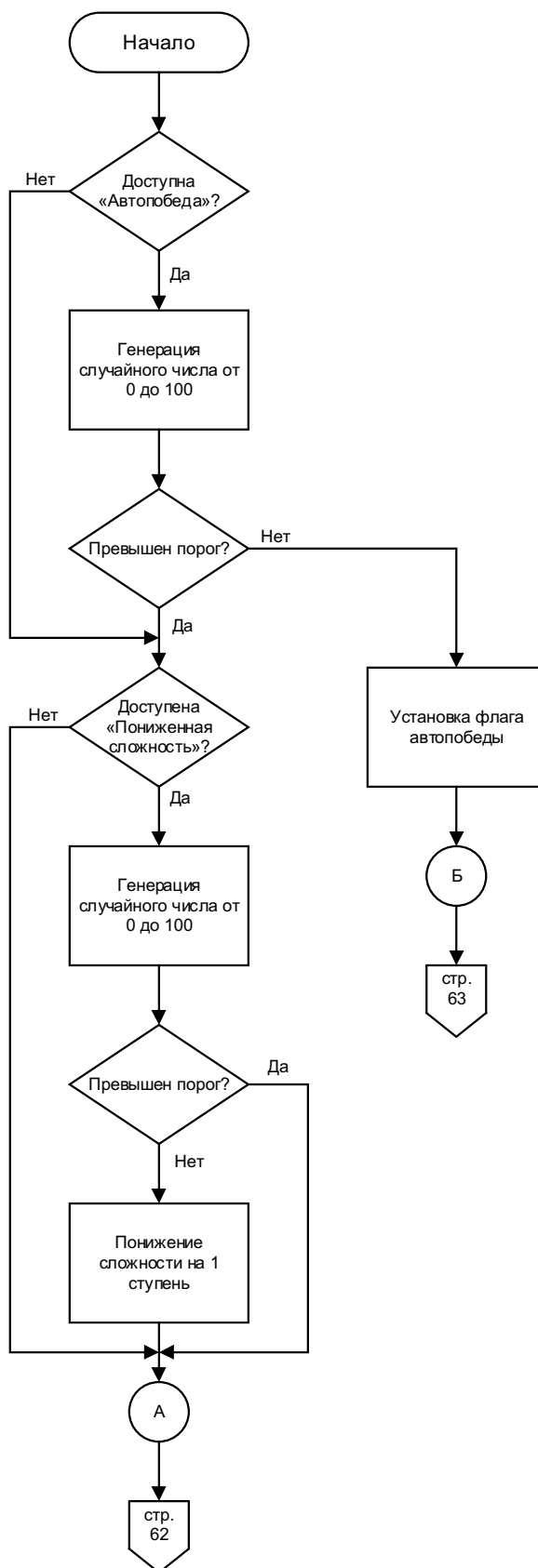
Блок-схема алгоритма генерации стартовых позиций для игры-головоломки «Викторина»

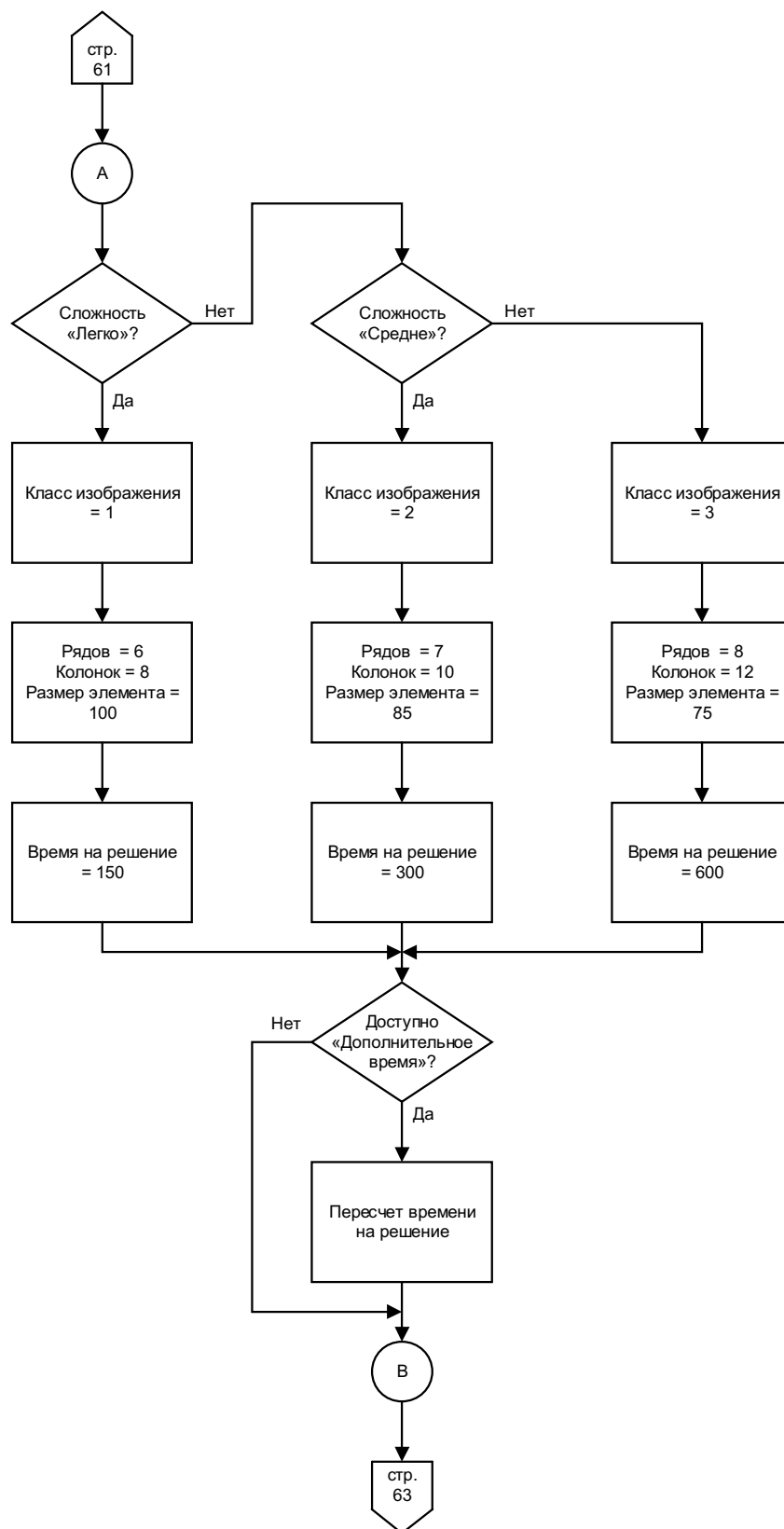


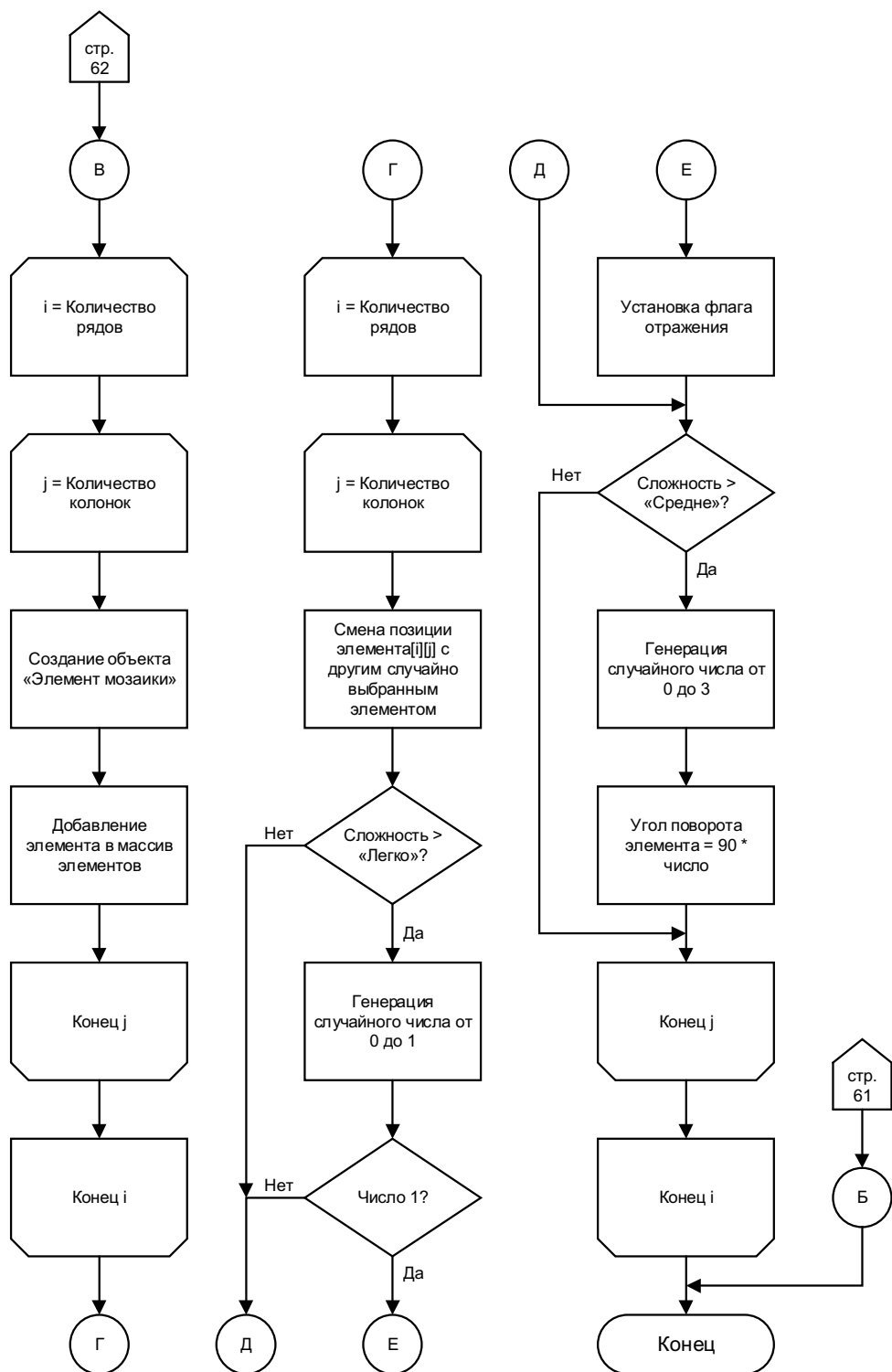


ПРИЛОЖЕНИЕ А3

Блок-схема алгоритма генерации стартовых позиций для игры-головоломки «Мозаика»

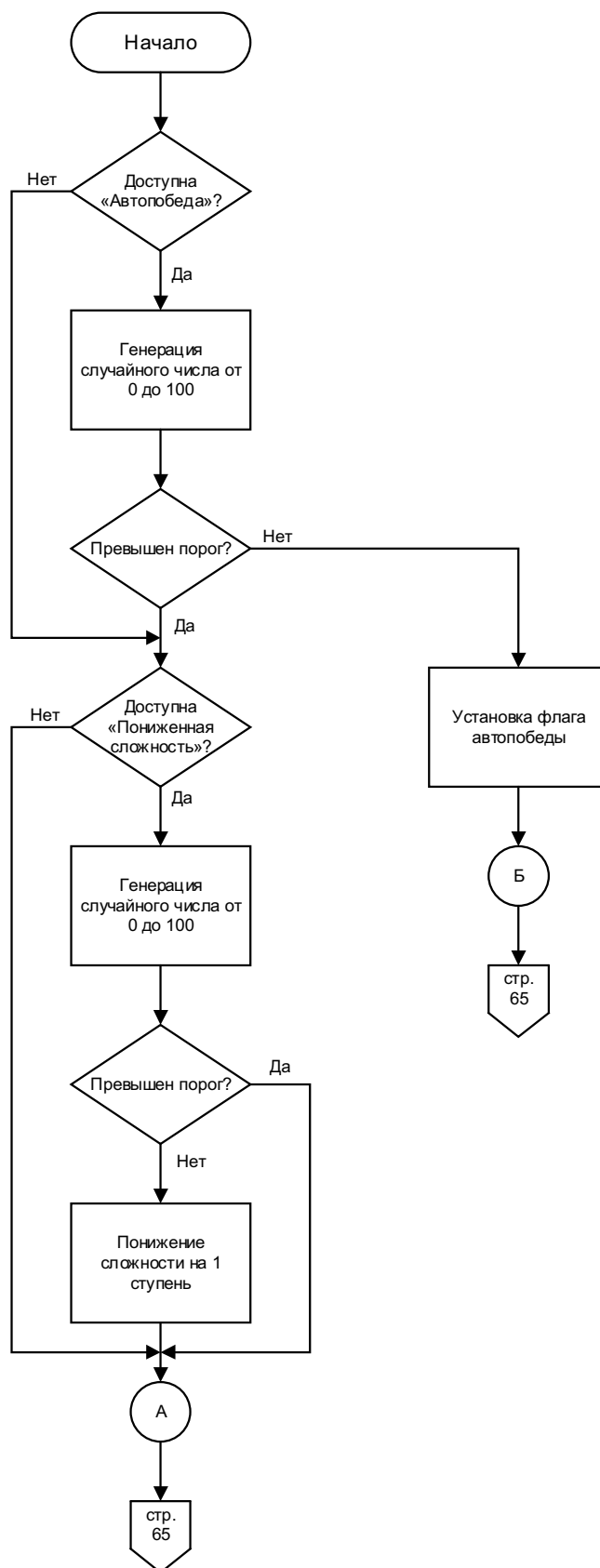


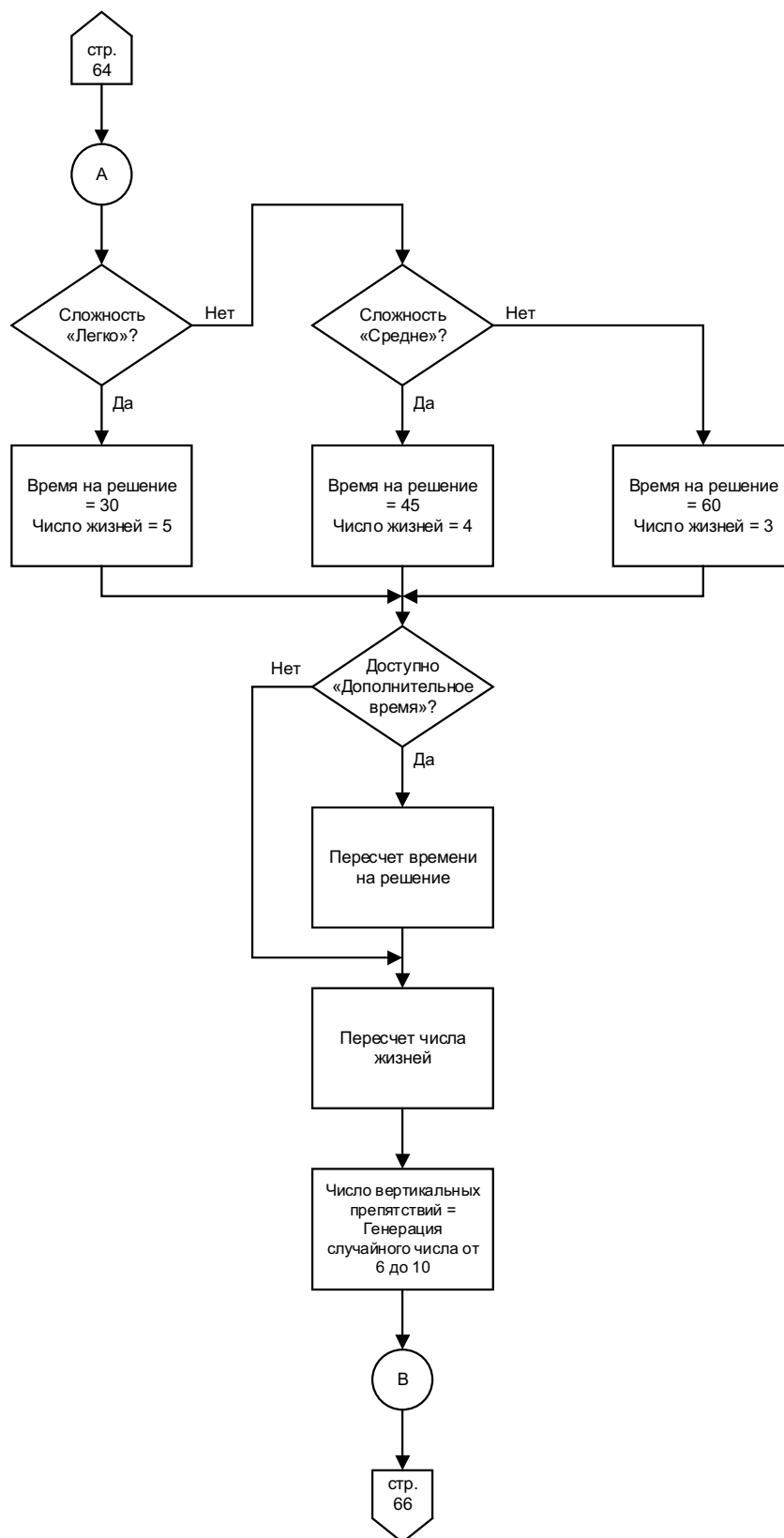


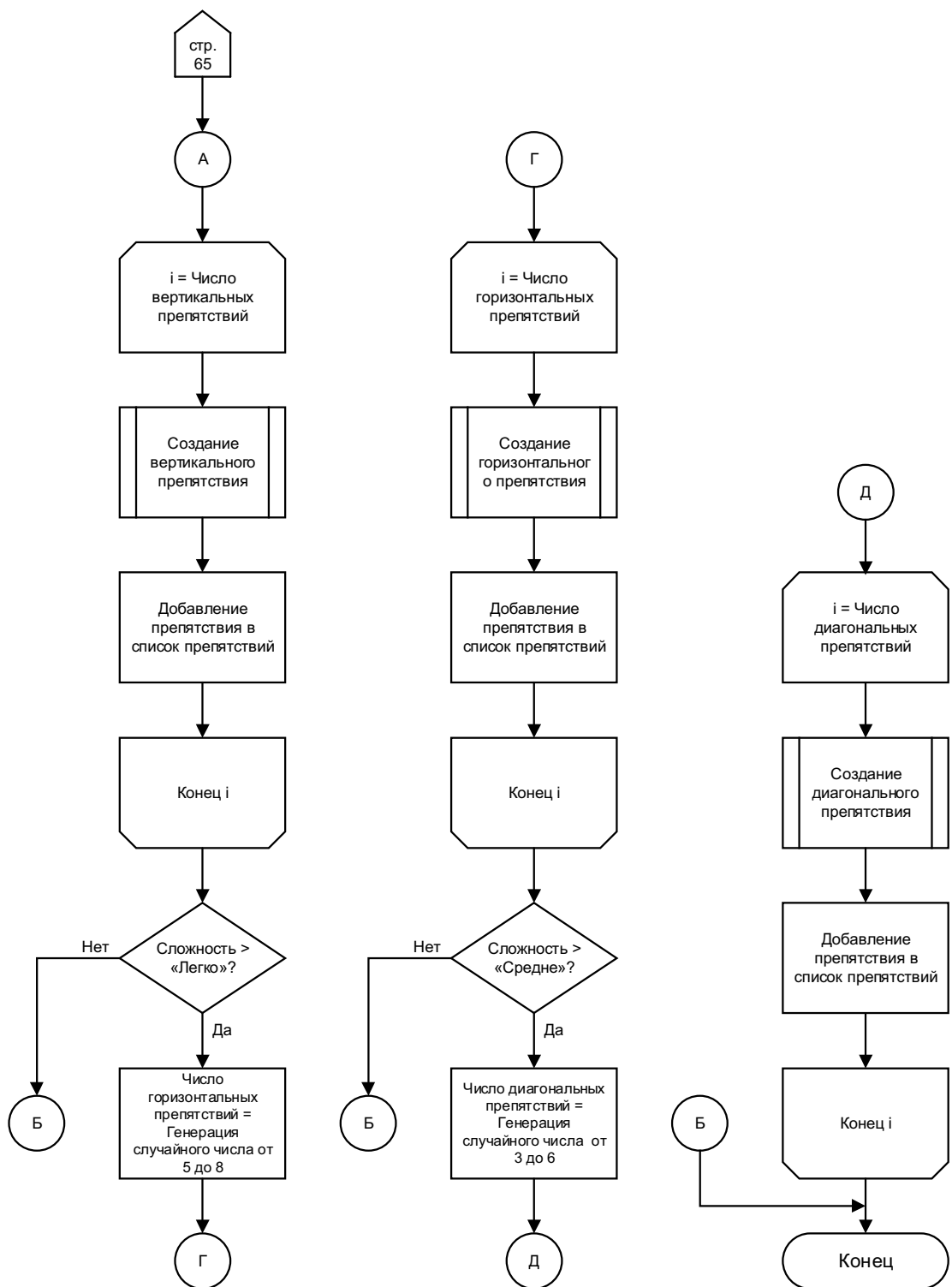


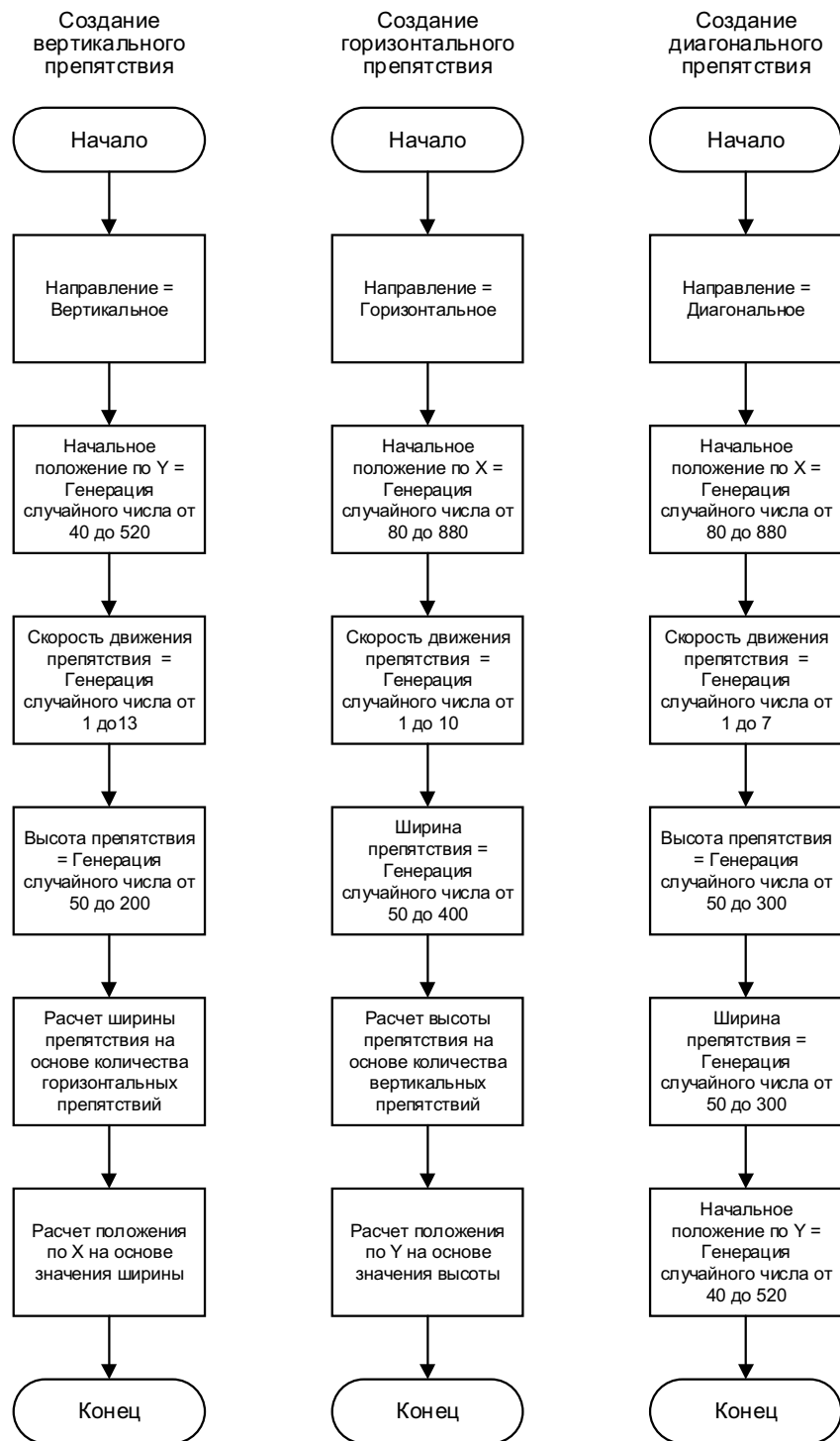
ПРИЛОЖЕНИЕ А4

Блок-схема алгоритма генерации стартовых позиций для игры-головоломки «Полоса препятствий»



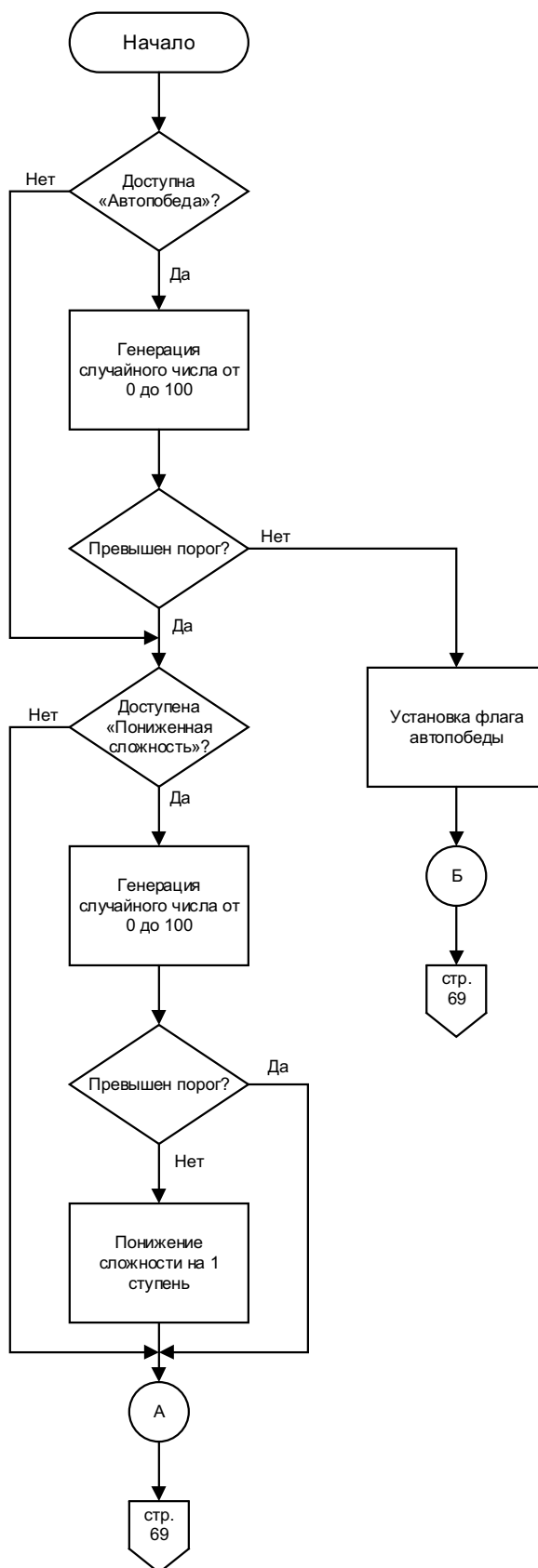


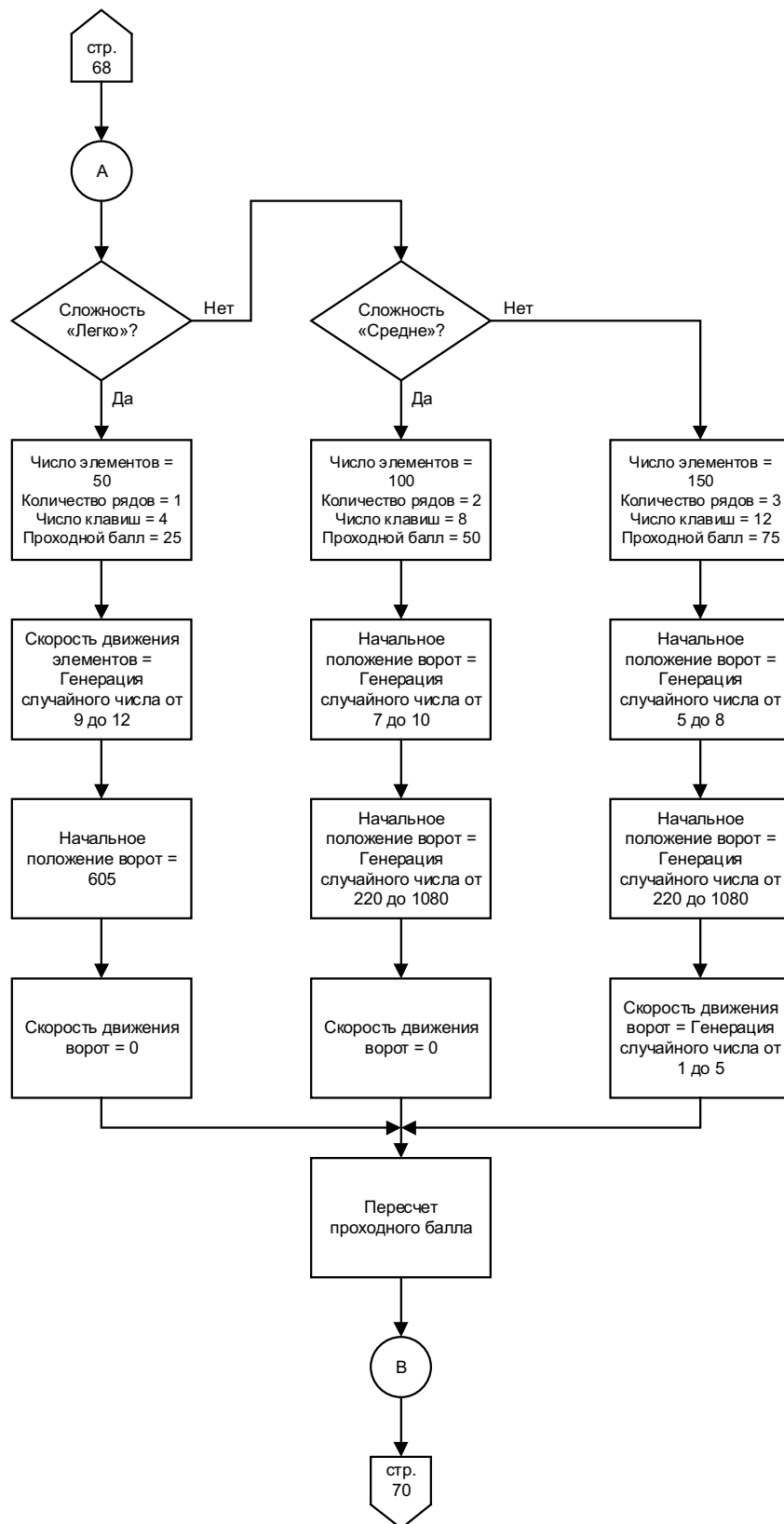




ПРИЛОЖЕНИЕ А5

Блок-схема алгоритма генерации стартовых позиций для игры-головоломки «Qte-испытание»

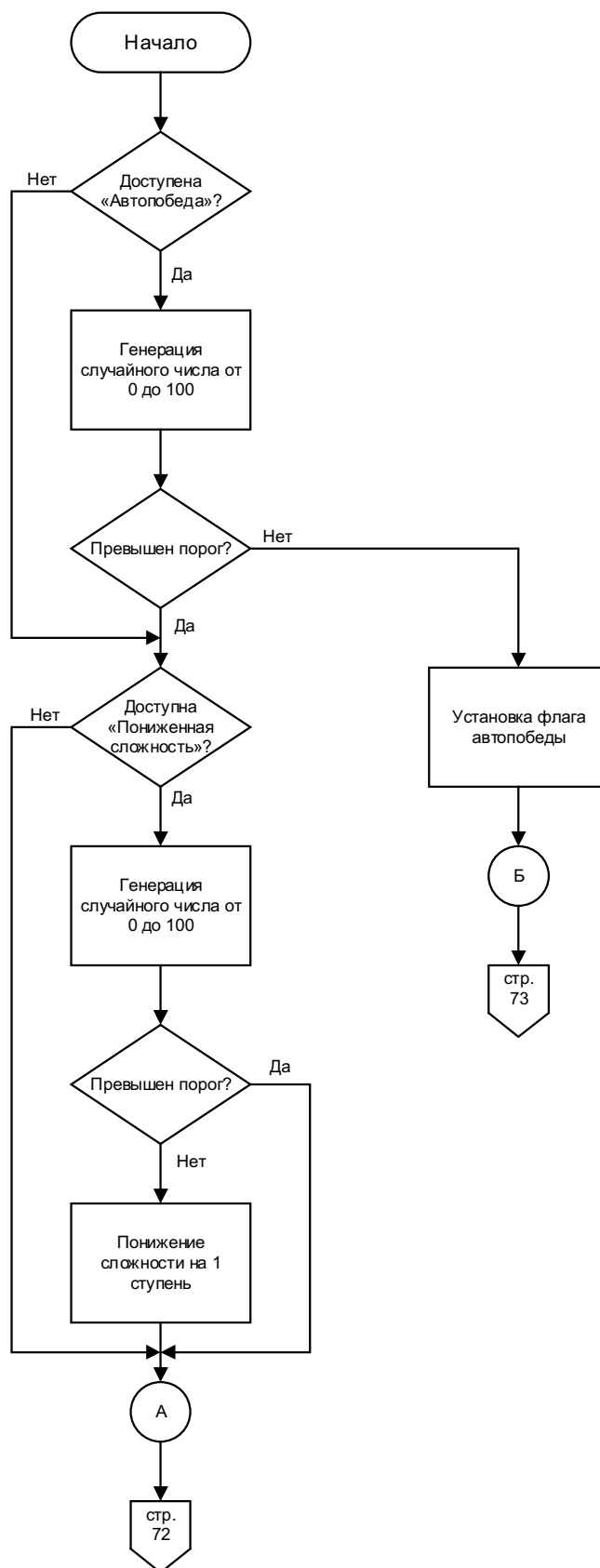


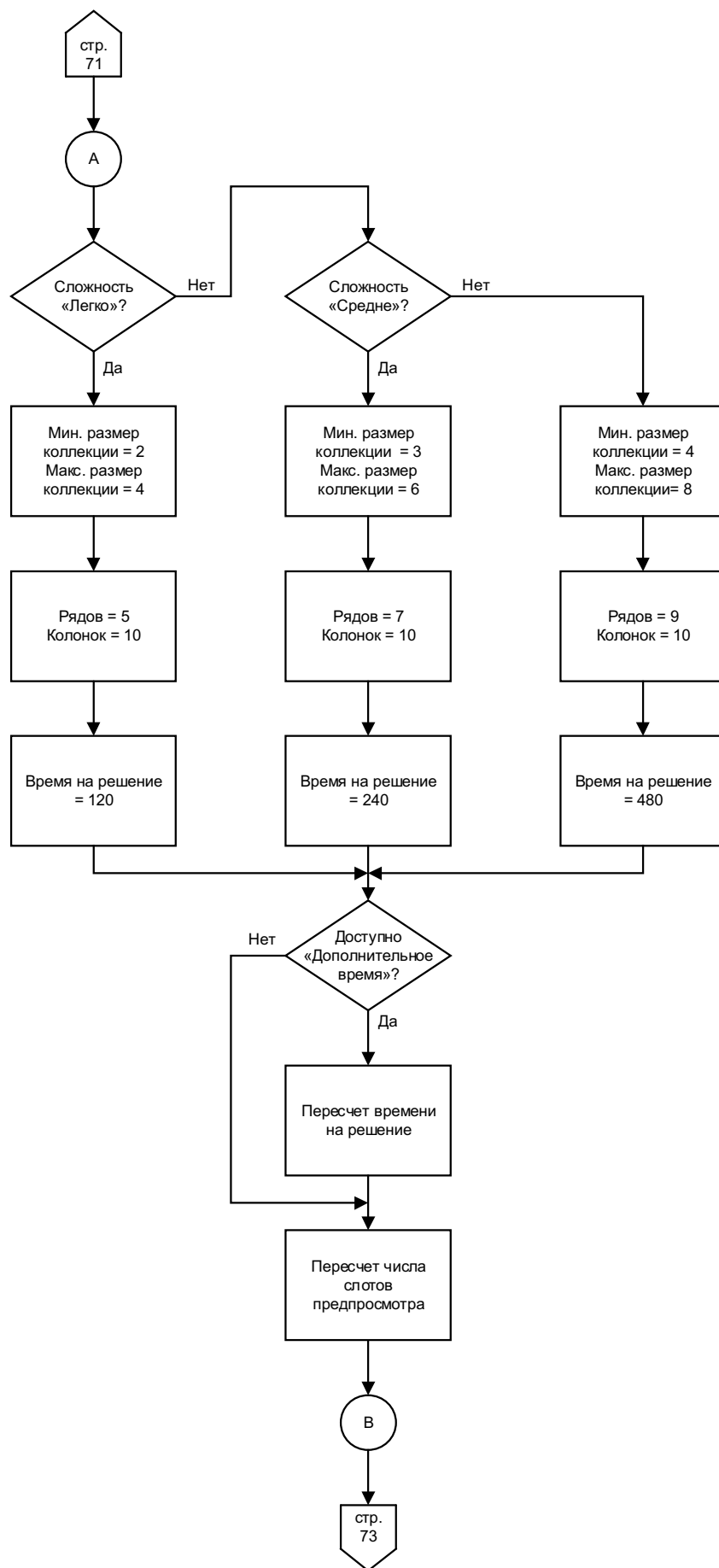


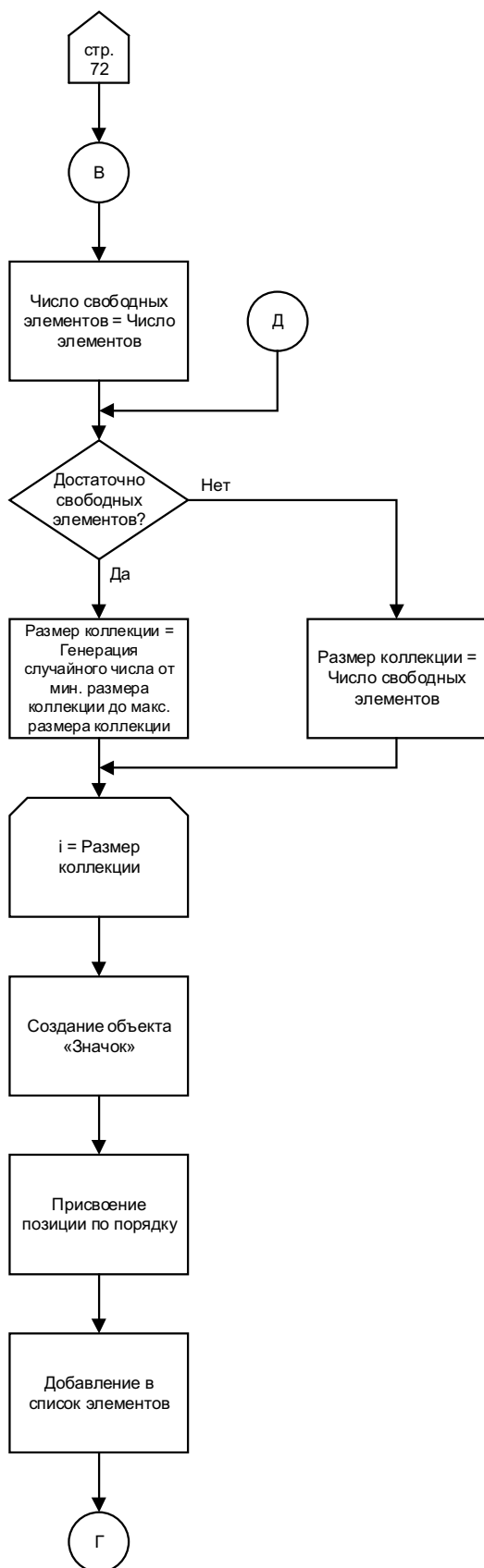


ПРИЛОЖЕНИЕ А6

Блок-схема алгоритма генерации стартовых позиций для игры-головоломки «Пары».







ПРИЛОЖЕНИЕ Б1

Листинг функции генерации точек входа

```
/// <summary>
/// Функция генерации точек входа
/// </summary>
/// <param name="entryPoints">Список точек входа</param>
/// <param name="activeSubjects">Список активных предметов</param>
public void GenerateEntryPoints(
    List<EntryPoint> entryPoints,
    ref List<Subject> activeSubjects )
{
    // Для всех предметов, которым требуется регенерация,
    // освобождаем занятые ими с предыдущей генерации аудитории
    foreach ( var ActiveSubject in activeSubjects )
    {
        if ( ActiveSubject.GenStatus == GenerationStatus.QuestRegeneration )
        {
            foreach ( var Ep in entryPoints )
            {
                if ( Ep.Name == ActiveSubject.AuditoriumName )
                {
                    Ep.IsBusy = false;
                }
            }
        }
    }

    // В зависимости от статуса флагов предмета определяем,
    // нужна ли предмету начальная генерация
    // или только регенерация задания
    foreach ( var ActiveSubject in activeSubjects )
    {
        // Если предмет нуждается в полной генерации
        if ( ActiveSubject.GenStatus == GenerationStatus.InitGeneration )
        {
            // Определяем 3 мини-игры
            var Res = Operations.RandomArray( 3, 0, 3 );

            // Устанавливаем мини-игры предмету
            ActiveSubject.SetMiniGames(
                // Шанс выпадения первой мини-игры от 50 до 65 процентов
                ( MiniGameType )Res[ 0 ],
                ( UInt32 )m_RandomGenerator.Next( 50, 65 ),

                // Шанс выпадения второй мини-игры от 25 до 30 процентов
                ( MiniGameType )Res[ 1 ],
                ( UInt32 )m_RandomGenerator.Next( 25, 30 ),

                // Шанс выпадения третьей мини-игры - все остальное
                ( MiniGameType )Res[ 2 ] );
        }

        // Если предмет нуждается в регенерации задания
        if ( ActiveSubject.GenStatus == GenerationStatus.QuestRegeneration )
        {
            // Каждому предмету для следующего задания
            // определяется своя случайная аудитория
            // Выбираем из списка аудиторий все названия свободных аудиторий
            var SelectedNames = ( from Ep in entryPoints
                                  where Ep.PointType == EntryPointType.Auditorium
                                  && !Ep.IsBusy
                                  select Ep.Name ).ToList();

            // Удаляем повторяющиеся значения, так как точки входа
            // в некоторые аудитории присутствуют с 2-х сторон
            SelectedNames = SelectedNames.Distinct().ToList();

            // Из оставшихся выбирает случайную аудиторию

```

```

var SelectedName =
    SelectedNames[ m_RandomGenerator.Next( 0, SelectedNames.Count )];

// Устанавливаем аудиторию с таким именем в состояние "занята"
foreach ( var Ep in entryPoints )
{
    if ( Ep.Name == SelectedName )
    {
        Ep.IsBusy = true;
    }
}

// Устанавливаем предмету имя аудитории, в котором
// будет происходить выполнение следующего задания
ActiveSubject.SetAuditorium( SelectedName );
}
}
}

```

ПРИЛОЖЕНИЕ Б2

Листинг функции генерации стартовой позиции для игры-головоломки

«Викторина»

```
/// <summary>
/// Функция генерации стартовой позиции для игры-головоломки "Викторина"
/// </summary>
/// <param name="difficulty">Сложность</param>
/// <returns>Конфигурация стартовой позиции</returns>
public QuizStartConfig GenerateQuizStart( GameDifficulty difficulty )
{
    var StartConfig = new QuizStartConfig( );

    var AutoWin =
        GameSession.Session.Attributes.Talents.GetTalent( "Автопобеда" );

    // Если доступен навык "Автопобеда"
    if ( AutoWin.Enabled )
    {
        var Chance = m_RandomGenerator.Next( 0, 100 );
        StartConfig.AutoWin = ( Chance <= ( Int32 )AutoWin.Value );
    }

    // Если сработала "Автопобеда" - завершаем генерацию
    if ( StartConfig.AutoWin )
        return StartConfig;

    var ReducedDifficulty =
        GameSession.Session.Attributes.Talents.GetTalent( "Пониженная сложность" );

    // Если доступен навык "Пониженная сложность"
    if ( ReducedDifficulty.Enabled )
    {
        var Chance = m_RandomGenerator.Next( 0, 100 );

        // Если сработала "Пониженная сложность"
        if ( Chance <= ( Int32 )ReducedDifficulty.Value )
        {
            StartConfig.Difficulty = ( difficulty == 0 ? difficulty : difficulty - 1 );
        }
    }

    // В соответствии со сложностью определяем динамические факторы
    switch ( difficulty )
    {
        case GameDifficulty.Easy:
            StartConfig.RemainingTime = new TimeSpan( 0, 2, 0 );
            StartConfig.QuestionsCount = m_RandomGenerator.Next( 5, 10 );
            StartConfig.AnswersCount = m_RandomGenerator.Next( 2, 4 );
            break;

        case GameDifficulty.Medium:
            StartConfig.RemainingTime = new TimeSpan( 0, 3, 0 );
            StartConfig.QuestionsCount = m_RandomGenerator.Next( 7, 15 );
            StartConfig.AnswersCount = m_RandomGenerator.Next( 3, 6 );
            break;

        case GameDifficulty.Hard:
            StartConfig.RemainingTime = new TimeSpan( 0, 4, 0 );
            StartConfig.QuestionsCount = m_RandomGenerator.Next( 10, 20 );
            StartConfig.AnswersCount = m_RandomGenerator.Next( 4, 8 );
            break;
    }

    var AdditionalTime =
        GameSession.Session.Attributes.Talents.GetTalent( "Дополнительное время" );
}
```

```

// Если доступен навык "Дополнительное время"
if ( AdditionalTime.Enabled )
{
    var Chance = m_RandomGenerator.Next( 0, 100 );
    if ( Chance <= ( Int32 )AdditionalTime.Value )
    {
        // Пересчитываем время
        StartConfig.RemainingTime = TimeSpan.FromSeconds(
            StartConfig.RemainingTime.TotalSeconds +
            StartConfig.RemainingTime.TotalSeconds *
            ( GameSession.Session.Attributes.Stamina / 100 * 3 ) );
    }
}

return StartConfig;
}

```

ПРИЛОЖЕНИЕ Б3

Листинг функции генерации стартовой позиции для игры-головоломки

«Мозаика»

```
/// <summary>
/// Функция генерации стартовой позиции для игры-головоломки "Мозаика"
/// </summary>
/// <param name="difficulty">Сложность</param>
/// <returns>Конфигурация стартовой позиции</returns>
public PuzzleStartConfig GeneratePuzzleStart( GameDifficulty difficulty )
{
    var StartConfig = new PuzzleStartConfig( );

    var AutoWin =
        GameSession.Session.Attributes.Talents.GetTalent( "Автопобеда" );

    // Если доступен навык "Автопобеда"
    if ( AutoWin.Enabled )
    {
        var Chance = m_RandomGenerator.Next( 0, 100 );
        StartConfig.AutoWin = ( Chance <= ( Int32 )AutoWin.Value );
    }

    // Если сработала "Автопобеда" - завершаем генерацию
    if ( StartConfig.AutoWin )
        return StartConfig;

    var ReducedDifficulty =
        GameSession.Session.Attributes.Talents.GetTalent( "Пониженная сложность" );

    // Если доступен навык "Пониженная сложность"
    if ( ReducedDifficulty.Enabled )
    {
        var Chance = m_RandomGenerator.Next( 0, 100 );

        // Если сработала "Пониженная сложность"
        if ( Chance <= ( Int32 )ReducedDifficulty.Value )
        {
            StartConfig.Difficulty = ( difficulty == 0 ? difficulty : difficulty - 1 );
        }
    }

    // В соответствии со сложностью определяем динамические факторы
    switch ( difficulty )
    {
        case GameDifficulty.Easy:
            StartConfig.Rows = 6;
            StartConfig.Columns = 8;
            StartConfig.ElementSize = 100;
            StartConfig.RemainingTime = new TimeSpan( 0, 2, 30 );
            break;

        case GameDifficulty.Medium:
            StartConfig.Rows = 6;
            StartConfig.Columns = 8;
            StartConfig.ElementSize = 100;
            StartConfig.RemainingTime = new TimeSpan( 0, 5, 0 );
            break;

        case GameDifficulty.Hard:
            StartConfig.Rows = 6;
            StartConfig.Columns = 8;
            StartConfig.ElementSize = 100;
            StartConfig.RemainingTime = new TimeSpan( 0, 10, 0 );
            break;
    }
}
```

```

var AdditionalTime =
    GameSession.Session.Attributes.Talents.GetTalent( "Дополнительное время" );

// Если доступен навык "Дополнительное время"
if ( AdditionalTime.Enabled )
{
    var Chance = m_RandomGenerator.Next( 0, 100 );
    if ( Chance <= ( Int32 )AdditionalTime.Value )
    {
        // Пересчитываем время
        StartConfig.RemainingTime = TimeSpan.FromSeconds(
            StartConfig.RemainingTime.TotalSeconds +
            StartConfig.RemainingTime.TotalSeconds *
            ( GameSession.Session.Attributes.Stamina / 100 * 3 ) );
    }
}

// Инициализация элементов мозаики
var Elements = new PuzzleElement[ StartConfig.Rows, StartConfig.Columns ];

for ( Int32 i = 0; i < StartConfig.Rows; i++ )
{
    for ( Int32 j = 0; j < StartConfig.Columns; j++ )
    {
        // При инициализации позиция каждого элемента мозаики
        // соответствует позиции захватываемого фрагмента изображения
        var PuzzleElemPos =
            new Point( StartConfig.ElementSize * i , StartConfig.ElementSize * j );

        Elements[ i, j ] =
            new PuzzleElement( PuzzleElemPos, PuzzleElemPos, StartConfig.ElementSize );
    }
}

// Перемешивание элементов, отражение, поворачивание
for ( Int32 i = 0; i < StartConfig.Rows; i++ )
{
    for ( Int32 j = 0; j < StartConfig.Columns; j++ )
    {
        var SwapI = m_RandomGenerator.Next( 0, StartConfig.Rows );
        var SwapJ = m_RandomGenerator.Next( 0, StartConfig.Columns );

        // Смена позиции текущего элемента и другого случайно выбранного элемента
        Elements[ i, j ].Swap( Elements[ SwapI, SwapJ ] );

        // Если сложность средняя или выше - добавляем отражение
        if ( StartConfig.Difficulty > GameDifficulty.Easy )
        {
            Elements[ i, j ].Reflection = ( m_RandomGenerator.Next( 0, 1 ) == 1 );
        }

        // Если сложность высокая - добавляем поворот
        if ( StartConfig.Difficulty > GameDifficulty.Medium )
        {
            Elements[ i, j ].Rotation = m_RandomGenerator.Next( 0, 3 );
        }
    }
}

// Преобразуем двумерный массив в список
StartConfig.Elements = Elements.Cast<PuzzleElement>( ).ToList( );

return StartConfig;
}

```

ПРИЛОЖЕНИЕ Б4

Листинг функции генерации стартовой позиции для игры-головоломки «Полоса препятствий»

```
/// <summary>
/// Функция генерации стартовой позиции для игры-головоломки "Полоса препятствий"
/// </summary>
/// <param name="difficulty">Сложность</param>
/// <returns>Конфигурация стартовой позиции</returns>
public ObstacleStartConfig GenerateObstacleStart( GameDifficulty difficulty )
{
    var StartConfig = new ObstacleStartConfig( );

    var AutoWin =
        GameSession.Session.Attributes.Talents.GetTalent( "Автопобеда" );

    // Если доступен навык "Автопобеда"
    if ( AutoWin.Enabled )
    {
        var Chance = m_RandomGenerator.Next( 0, 100 );
        StartConfig.AutoWin = ( Chance <= ( Int32 )AutoWin.Value );
    }

    // Если сработала "Автопобеда" - завершаем генерацию
    if ( StartConfig.AutoWin )
        return StartConfig;

    var ReducedDifficulty =
        GameSession.Session.Attributes.Talents.GetTalent( "Пониженная сложность" );

    // Если доступен навык "Пониженная сложность"
    if ( ReducedDifficulty.Enabled )
    {
        var Chance = m_RandomGenerator.Next( 0, 100 );

        // Если сработала "Пониженная сложность"
        if ( Chance <= ( Int32 )ReducedDifficulty.Value )
        {
            StartConfig.Difficulty = ( difficulty == 0 ? difficulty : difficulty - 1 );
        }
    }

    // В соответствии со сложностью определяем динамические факторы
    switch ( difficulty )
    {
        case GameDifficulty.Easy:
            StartConfig.RemainingTime = new TimeSpan( 0, 0, 30 );
            StartConfig.LivesCount = 5;
            break;

        case GameDifficulty.Medium:
            StartConfig.RemainingTime = new TimeSpan( 0, 0, 45 );
            StartConfig.LivesCount = 4;
            break;

        case GameDifficulty.Hard:
            StartConfig.RemainingTime = new TimeSpan( 0, 1, 0 );
            StartConfig.LivesCount = 3;
            break;
    }

    var AdditionalTime =
        GameSession.Session.Attributes.Talents.GetTalent( "Дополнительное время" );

    // Если доступен навык "Дополнительное время"
    if ( AdditionalTime.Enabled )
    {
```

```

var Chance = m_RandomGenerator.Next( 0, 100 );
if ( Chance <= ( Int32 )AdditionalTime.Value )
{
    // Пересчитываем время
    StartConfig.RemainingTime = TimeSpan.FromSeconds(
        StartConfig.RemainingTime.TotalSeconds +
        StartConfig.RemainingTime.TotalSeconds *
        ( GameSession.Session.Attributes.Stamina / 100 * 3 ) );
}
}

// Пересчет числа жизней
StartConfig.LivesCount += ( Int32 )Math.Ceiling(
    ( double ) ( GameSession.Session.Attributes.Stamina / 10 ) );

StartConfig.Obstacles = new List<ObstacleObj>( );

// Определение числа вертикальных препятствий
var VerticalCount = m_RandomGenerator.Next( 6, 10 );

for ( Int32 i = 0; i < VerticalCount; i++ )
{
    // Создаем вертикальный объект и добавляем в список
    StartConfig.Obstacles.Add( GenerateVerticleObstacle( i, VerticalCount ) );
}

if ( StartConfig.Difficulty > GameDifficulty.Easy )
{
    // Определение числа горизонтальных препятствий
    var HorizontalCount = m_RandomGenerator.Next( 5, 8 );

    for ( Int32 i = 0; i < HorizontalCount; i++ )
    {
        // Создаем горизонтальный объект и добавляем в список
        StartConfig.Obstacles.Add( GenerateHorizontalObstacle( i, HorizontalCount ) );
    }
}

if ( StartConfig.Difficulty > GameDifficulty.Medium )
{
    // Определение числа диагональных препятствий
    var DiagonalCount = m_RandomGenerator.Next( 3, 6 );

    for ( Int32 i = 0; i < DiagonalCount; i++ )
    {
        // Создаем диагональный объект и добавляем в список
        StartConfig.Obstacles.Add( GenerateDiagonalObstacle( i, DiagonalCount ) );
    }
}

return StartConfig;
}

/// <summary>
/// Генерация одного вертикального препятствия
/// </summary>
/// <param name="current">Порядковый номер препятствия</param>
/// <param name="total">Общее число вертикальных препятствий</param>
/// <returns>Вертикальное препятствие</returns>
private ObstacleObj GenerateVerticleObstacle( Int32 current, Int32 total )
{
    // Определение направления движения
    var Direction = ObstacleDirection.Vertical;

    // Генерация позиции по координате Y
    var YPos = m_RandomGenerator.Next( 40, 520 );

    // Генерация значения скорости
    var VSpeed = m_RandomGenerator.Next( 1, 13 );

```

```

// Генерация размера по высоте
var Height = m_RandomGenerator.Next( 50, 200 );

// Определяем ширину площади, по которой будет двигаться 1 элемент
// для того, чтобы равномерно распределить их по всей площади
var ObstacleAreaWidth = ( Int32 )Math.Ceiling( ( double )1120 / total );

// Исходя из этого определяем ширину препятствия
var Width = m_RandomGenerator.Next(
    ( Int32 )Math.Ceiling( ObstacleAreaWidth * 0.5 ),
    ( Int32 )Math.Ceiling( ObstacleAreaWidth * 0.8 ) );

// Исходя из значения ширины, определяем позицию
// препятствия по координате X с учетом отступа в 80
var XPos = ( ObstacleAreaWidth - Width ) / 2 + current * ObstacleAreaWidth + 40;

// Создаем препятствие
return new ObstacleObj(
    new Rectangle( XPos, YPos, Height, Width ),
    Direction,
    VSpeed );
}

/// <summary>
/// Генерация одного горизонтального препятствия
/// </summary>
/// <param name="current">Порядковый номер препятствия</param>
/// <param name="total">Общее число горизонтальных препятствий</param>
/// <returns>Горизонтальное препятствие</returns>
private ObstacleObj GenerateHorizontalObstacle( Int32 current, Int32 total )
{
    // Определение направления движения
    var Direction = ObstacleDirection.Horizontal;

    // Генерация позиции по координате X
    var XPos = m_RandomGenerator.Next( 80, 880 );

    // Генерация значения скорости
    var HSpeed = m_RandomGenerator.Next( 1, 10 );

    // Генерация размера по ширине
    var Width = m_RandomGenerator.Next( 50, 400 );

    // Определяем высоту площади, по которой будет двигаться 1 элемент
    // для того, чтобы равномерно распределить их по всей площади
    var ObstacleAreaHeight = ( Int32 )Math.Ceiling( ( double )640 / total );

    // Исходя из этого определяем высоту препятствия
    var Height = m_RandomGenerator.Next(
        ( Int32 )Math.Ceiling( ObstacleAreaHeight * 0.5 ),
        ( Int32 )Math.Ceiling( ObstacleAreaHeight * 0.8 ) );

    // Исходя из значения высоты, определяем позицию
    // препятствия по координате Y с учетом отступа в 80
    var YPos = ( ObstacleAreaHeight - Height ) / 2 + current * ObstacleAreaHeight + 80;

    // Создаем препятствие
    return new ObstacleObj(
        new Rectangle( XPos, YPos, Height, Width ),
        Direction,
        HSpeed );
}

/// <summary>
/// Генерация одного диагонального препятствия
/// </summary>
/// <param name="current">Порядковый номер препятствия</param>
/// <param name="total">Общее число диагональных препятствий</param>
/// <returns>Диагональное препятствие</returns>
private ObstacleObj GenerateDiagonalObstacle( Int32 current, Int32 total )

```

```

{
    // Определение направления движения
    var Direction = ObstacleDirection.Diagonal;

    // Генерация позиции по координате Y
    var YPos = m_RandomGenerator.Next( 40, 520 );

    // Генерация позиции по координате X
    var XPos = m_RandomGenerator.Next( 80, 880 );

    // Генерация значения скорости
    var HSpeed = m_RandomGenerator.Next( 1, 13 );
    var VSpeed = m_RandomGenerator.Next( 1, 13 );

    // Генерация размера по высоте
    var Height = m_RandomGenerator.Next( 50, 200 );

    // Генерация размера по ширине
    var Width = m_RandomGenerator.Next( 50, 300 );

    // Создаем препятствие
    return new ObstacleObj(
        new Rectangle( XPos, YPos, Height, Width ),
        Direction,
        HSpeed );
}

```

ПРИЛОЖЕНИЕ Б5

Листинг функции генерации стартовой позиции для игры-головоломки «Qte-испытание»

```
/// <summary>
/// Функция генерации стартовой позиции для игры-головоломки "Qte-испытание"
/// </summary>
/// <param name="difficulty">Сложность</param>
/// <returns>Конфигурация стартовой позиции</returns>
public QteObstacleStartConfig GenerateQteObstacleStart( GameDifficulty difficulty )
{
    var StartConfig = new QteObstacleStartConfig( );

    var AutoWin =
        GameSession.Session.Attributes.Talents.GetTalent( "Автопобеда" );

    // Если доступен навык "Автопобеда"
    if ( AutoWin.Enabled )
    {
        var Chance = m_RandomGenerator.Next( 0, 100 );
        StartConfig.AutoWin = ( Chance <= ( Int32 )AutoWin.Value );
    }

    // Если сработала "Автопобеда" - завершаем генерацию
    if ( StartConfig.AutoWin )
        return StartConfig;

    var ReducedDifficulty =
        GameSession.Session.Attributes.Talents.GetTalent( "Пониженная сложность" );

    // Если доступен навык "Пониженная сложность"
    if ( ReducedDifficulty.Enabled )
    {
        var Chance = m_RandomGenerator.Next( 0, 100 );

        // Если сработала "Пониженная сложность"
        if ( Chance <= ( Int32 )ReducedDifficulty.Value )
        {
            StartConfig.Difficulty = ( difficulty == 0 ? difficulty : difficulty - 1 );
        }
    }

    // В соответствии со сложностью определяем динамические факторы
    switch ( difficulty )
    {
        case GameDifficulty.Easy:
            StartConfig.ElementsCount = 50;
            StartConfig.RowsCount = 1;
            StartConfig.KeysCount = 4;
            StartConfig.PassingScore = 25;
            StartConfig.ElementSpeed = m_RandomGenerator.Next( 9, 12 );
            StartConfig.GatePosition = 605;
            StartConfig.GateSpeed = 0;
            break;

        case GameDifficulty.Medium:
            StartConfig.ElementsCount = 100;
            StartConfig.RowsCount = 2;
            StartConfig.KeysCount = 8;
            StartConfig.PassingScore = 50;
            StartConfig.ElementSpeed = m_RandomGenerator.Next( 7, 10 );
            StartConfig.GatePosition = m_RandomGenerator.Next( 220, 180 );
            StartConfig.GateSpeed = 0;
            break;

        case GameDifficulty.Hard:
            StartConfig.ElementsCount = 150;
```

```

        StartConfig.RowsCount = 3;
        StartConfig.KeysCount = 12;
        StartConfig.PassingScore = 75;
        StartConfig.ElementSpeed = m_RandomGenerator.Next( 5, 8 );
        StartConfig.GatePosition = m_RandomGenerator.Next( 220, 180 );
        StartConfig.GateSpeed = m_RandomGenerator.Next( 1, 5 );
        break;
    }

    // Пересчет проходного балла
    StartConfig.PassingScore = ( Int32 )Math.Ceiling(
        StartConfig.PassingScore *
        ( double )( 1 - GameSession.Session.Attributes.Intelligence / 200 ) );

    // Создаем элементы
    for ( Int32 i = 0; i < StartConfig.ElementsCount; i++ )
    {
        // Генерация клавиши
        var Key = m_RandomGenerator.Next( 1, StartConfig.KeysCount );

        // Генерация ряда
        var Row = m_RandomGenerator.Next( 1, StartConfig.RowsCount );

        // Создание элемента и добавление в ряд
        StartConfig.Elements.Add( new QteElement( Key, Row, StartConfig.ElementSpeed ) );
    }

    return StartConfig;
}

```

ПРИЛОЖЕНИЕ Б6

Листинг функции генерации стартовой позиции для игры-головоломки

«Пары»

```
/// <summary>
/// Функция генерации стартовой позиции для игры-головоломки "Пары"
/// </summary>
/// <param name="difficulty">Сложность</param>
/// <returns>Конфигурация стартовой позиции</returns>
public IconPairsStartConfig GenerateIconPairsStart( GameDifficulty difficulty )
{
    var StartConfig = new IconPairsStartConfig( );

    var AutoWin =
        GameSession.Session.Attributes.Talents.GetTalent( "Автопобеда" );

    // Если доступен навык "Автопобеда"
    if ( AutoWin.Enabled )
    {
        var Chance = m_RandomGenerator.Next( 0, 100 );
        StartConfig.AutoWin = ( Chance <= ( Int32 )AutoWin.Value );
    }

    // Если сработала "Автопобеда" - завершаем генерацию
    if ( StartConfig.AutoWin )
        return StartConfig;

    var ReducedDifficulty =
        GameSession.Session.Attributes.Talents.GetTalent( "Пониженная сложность" );

    // Если доступен навык "Пониженная сложность"
    if ( ReducedDifficulty.Enabled )
    {
        var Chance = m_RandomGenerator.Next( 0, 100 );

        // Если сработала "Пониженная сложность"
        if ( Chance <= ( Int32 )ReducedDifficulty.Value )
        {
            StartConfig.Difficulty = ( difficulty == 0 ? difficulty : difficulty - 1 );
        }
    }

    // В соответствии со сложностью определяем динамические факторы
    switch ( difficulty )
    {
        case GameDifficulty.Easy:
            StartConfig.CollectionMinSize = 2;
            StartConfig.CollectionMaxSize = 4;
            StartConfig.PreviewSlotsCount = 4;
            StartConfig.Rows = 5;
            StartConfig.Columns = 10;
            StartConfig.RemainingTime = new TimeSpan( 0, 2, 0 );
            break;

        case GameDifficulty.Medium:
            StartConfig.CollectionMinSize = 3;
            StartConfig.CollectionMaxSize = 6;
            StartConfig.PreviewSlotsCount = 6;
            StartConfig.Rows = 7;
            StartConfig.Columns = 10;
            StartConfig.RemainingTime = new TimeSpan( 0, 4, 0 );
            break;

        case GameDifficulty.Hard:
            StartConfig.CollectionMinSize = 4;
            StartConfig.CollectionMaxSize = 8;
            StartConfig.PreviewSlotsCount = 8;
```

```

        StartConfig.Rows = 9;
        StartConfig.Columns = 10;
        StartConfig.RemainingTime = new TimeSpan( 0, 8, 0 );
        break;
    }

    var AdditionalTime =
        GameSession.Session.Attributes.Talents.GetTalent( "Дополнительное время" );

    // Если доступен навык "Дополнительное время"
    if ( AdditionalTime.Enabled )
    {
        var Chance = m_RandomGenerator.Next( 0, 100 );
        if ( Chance <= ( Int32 )AdditionalTime.Value )
        {
            // Пересчитываем время
            StartConfig.RemainingTime = TimeSpan.FromSeconds(
                StartConfig.RemainingTime.TotalSeconds +
                StartConfig.RemainingTime.TotalSeconds *
                ( GameSession.Session.Attributes.Stamina / 100 * 3 ) );
        }
    }

    // Пересчет числа слотов предпросмотра
    StartConfig.PreviewSlotsCount += ( Int32 )Math.Ceiling(
        ( double )StartConfig.PreviewSlotsCount *
        GameSession.Session.Attributes.Intelligence / 100 );

    // Генерация коллекций
    var FreeElements = StartConfig.Rows * StartConfig.Columns;

    var PositionPoint = new Point( 0, 0 );

    var Elements = new List<Icon>( );

    Int32 CollectionId = 1;

    // Пока не проинициализируем все переменные
    while ( FreeElements > 0 )
    {
        var CollectionSize = 0;

        // Если число элементов больше максимального размера коллекции
        if ( FreeElements > StartConfig.CollectionMaxSize )
        {
            // Генерируем размер коллекции
            CollectionSize = m_RandomGenerator.Next(
                StartConfig.CollectionMinSize, StartConfig.CollectionMaxSize );
        }
        else
        {
            CollectionSize = FreeElements;
        }

        // Инициализация элементов
        for ( Int32 i = 0; i < CollectionSize; i++ )
        {
            // Изначально элементы размещаются по порядку
            Elements.Add( new Icon( PositionPoint, CollectionId ) );

            if ( PositionPoint.X < StartConfig.Columns )
            {
                PositionPoint.X++;
            }
            else
            {
                PositionPoint.X = 0;
                PositionPoint.Y++;
            }
        }
    }

```

```

        // Вычетаем занятые элементы
        FreeElements -= CollectionSize;
    }

    // Перемешиваем элементы
    foreach ( var IconElement in Elements )
    {
        IconElement.Swap( Elements[ m_RandomGenerator.Next( 0, Elements.Count ) ] );
    }

    StartConfig.Icons = Elements;

    return StartConfig;
}

```