**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное автономное образовательное   
учреждение высшего образования   
«Санкт-Петербургский политехнический университет Петра Великого»  
ИНСТИТУТ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

|  |  |
| --- | --- |
|  | **Допустить к защите** Заместитель директора  по УМР \_\_\_\_\_\_\_\_\_\_\_\_ Е.Г. Конакина\_  (Подпись) (ФИО)  «\_\_\_»\_\_\_\_\_\_\_\_\_\_ 2025 г. |
|  |

**курсовой проект**Тема: «Игра с графическим интерфейсом «Тетрис»

специальность 09.02.07 группа 32919/22



Студент (ка) Лелюков М.А.  
 (подпись) (ФИО)  
Преподаватель Девятко Н.С.

(подпись) (ФИО)

Санкт-Петербург  
2025

СОДЕРЖАНИЕ

[1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ 5](#_Toc207279249)

[1.1. Описание предметной области 5](#_Toc207279250)

[1.2. Анализ методов решения 8](#_Toc207279251)

[1.3. Обзор средств программирования 8](#_Toc207279252)

[1.4. Описание языка Golang 9](#_Toc207279253)

[2. ПРАКТИЧЕСКАЯ ЧАСТЬ 11](#_Toc207279254)

[2.1. Постановка задачи 11](#_Toc207279255)

[2.1.1. Основания для разработки 11](#_Toc207279256)

[2.1.2. Назначение программы 11](#_Toc207279257)

[2.2. Проектирование приложения 11](#_Toc207279258)

[2.2.1. Диаграмма прецедентов 12](#_Toc207279259)

[2.2.2. Диаграмма последовательностей 13](#_Toc207279260)

[2.2.3. Диаграмма классов 14](#_Toc207279261)

[2.2.4. Диаграмма активности 14](#_Toc207279262)

[2.3. Текст программы 15](#_Toc207279263)

[2.4. Описание программы 16](#_Toc207279264)

[2.4.1. Общие сведения 16](#_Toc207279265)

[2.4.2. Функциональное назначение 16](#_Toc207279266)

[2.4.3. Описание логической структуры системы 16](#_Toc207279267)

[2.4.4. Используемые технические и программные средства 18](#_Toc207279268)

[2.4.5. Вызов и загрузка 18](#_Toc207279269)

[2.5. Руководство оператора 19](#_Toc207279270)

[2.5.1. Назначение программы 19](#_Toc207279271)

[2.5.2. Выполнение программы и сообщения оператору 19](#_Toc207279272)

[2.6.1. Объект испытаний 20](#_Toc207279273)

[2.6.3. Требования к программе 20](#_Toc207279274)

[2.6.4. Требования к программной документации 22](#_Toc207279275)

[2.6.5. Средства и порядок испытаний 22](#_Toc207279276)

[2.6.6. Методы испытаний 23](#_Toc207279277)

[2.7. Протокол испытаний 25](#_Toc207279278)

[ЗАКЛЮЧЕНИЕ 27](#_Toc207279279)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 28](#_Toc207279280)

[ПРИЛОЖЕНИЕ А 29](#_Toc207279281)

[ПРИЛОЖЕНИЕ Б 37](#_Toc207279282)

[ПРИЛОЖЕНИЕ В 42](#_Toc207279283)

# 

# ВВЕДЕНИЕ

В настоящее время компьютеры широко применяются в самых разных областях: науке, промышленности, медицине, образовании. Применение компьютеров в системе образования значительно облегчает труд преподавателя, так как позволяет быстрее и легче проверить знания учащихся, подготовить для них обучающие и тестовые задания. Применение компьютеров способствует также повышению уровня знаний студентов. Большое количество игровых программ способствует повышению интереса пользователей к компьютерным технологиям. Многие игры носят не только развлекательный характер: они повышают скорость реакции, развивают память, внимание, логическое мышление, скорость принятия решения в нестандартных ситуациях.

Разработанная в данном курсовом проекте программа может быть использована в развлекательных целях, для отдыха и релаксации, а также для повышения общей реакции и концентрации.

# 

# 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ

## 1.1. Описание предметной области

Компьютерная игра́ (также — видеоигра) — компьютерная программа, служащая для организации игрового процесса (геймплея), связи с партнёрами по игре, или сама выступающая в качестве партнёра.

Компьютерные игры могут создаваться на основе фильмов и книг; есть и обратные случаи. С 2011 года компьютерные игры официально признаны в США отдельным видом искусства.

Компьютерные игры могут быть классифицированы по нескольким признакам:

**Жанр:** игра может принадлежать как к одному, так и к нескольким жанрам, а в уникальных случаях — открывать новый или быть вне всяких жанров;

**Количество игроков и способ их взаимодействия:** игра может быть однопользовательской — рассчитанной на игру одного человека, или многопользовательской — рассчитанной на одновременную игру нескольких человек; а также вестись на одном компьютере, через интернет, электронную почту, или массово;

**Визуальное представление:** игра может как использовать графические средства оформления, так и напротив, быть текстовой. Игра также может быть двухмерной или трехмерной, есть также графика 2,5D. В этом случае фоновое изображение и объекты на нем прорисованы как трехмерные, но персонаж движется в двухмерном пространстве. Есть и звуковые игры — в них вместо визуального представления используются звуки и голосовые команды.

**Платформа:** игра может принадлежать как к одной платформе, так и быть мультиплатформенной.

Жанр определяется целью игры. Выделяют следующие жанры:

* *Приключенческая игра* (Adventure) — игра, обладающая полноценным литературным сюжетом, и игрок в процессе игры сам раскрывает все перипетии этого сюжета.
* *Боевик* (Action) — игра, состоящая, в основном, из боевых сцен, драк и перестрелок. Боевики подразделяются на:
  + «шутеры» (от англ. *shoot — стрелять*) (пример: Counter-Strike, серии игр Call of Duty и Battlefield);
  + «файтинг» (от англ. *fight — драться*) (пример: Mortal Kombat, Street Fighter);
  + «ужасы» (примеры: серия игр F.E.A.R., Silent Hill, Resident Evil);
  + «стелс» (от англ. *stealth — скрытный*) (пример: серии игр Hitman, Metal Gear Solid и Splinter Cell).
* *Ролевая игра* (RPG — англ. *Role Playing Game*) — игра, отличительной особенностью которой является наличие у персонажей определённых навыков и характеристик, которые можно обрести, а впоследствии развивать, выполняя какие-либо действия. К этому жанру относятся и многопользовательские ролевые игры, которые, в отличие от однопользовательских, не имеют ни конечной цели, ни законченного сюжета. Примеры: The Elder Scrolls (серия), Gothic (серия)
* *Стратегическая игра* (Strategy) — игра, представляющая собой управление масштабными процессами, как, например, строительство городов, ведение бизнеса, командование армией и т. д. Игровой процесс может идти как в реальном времени (RTS — real time strategy), так и в пошаговом режиме (TBS — turn based strategy).
* *Компьютерный симулятор* (Simulator) — игра, полностью имитирующая какую-либо область реальной жизни, например, имитация управления гоночным автомобилем или самолётом.
* *Головоломка* (Puzzle) — игра, полностью или более чем наполовину состоящая из решения различных логических задач и головоломок.
* *Образовательная игра* — игра, включающая в себя элементы обучающих программ, которые подаются через сам игровой процесс и, благодаря повышению интереса к ним в связи с необычным антуражем, впоследствии хорошо запоминаются.
* *Забавы* — игры, рассчитанные, в основном, на детей, где психологическое впечатление от происходящего на экране гораздо важнее самого процесса игры — например, вид лопающихся пузырьков.

Жанровая классификация компьютерных игр предполагает введение нескольких дополнительных оснований:

* динамика — игровой процесс может происходить в условиях «реального времени» или пошагово;
* перспектива — игра может вестись как от первого, так и от третьего лица.

По некоторым компьютерным играм проводятся любительские и профессиональные [соревнования](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D0%B5%D0%B2%D0%BD%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) — такого рода соревнования называются [киберспортом](https://ru.wikipedia.org/wiki/%D0%9A%D0%B8%D0%B1%D0%B5%D1%80%D1%81%D0%BF%D0%BE%D1%80%D1%82). Игры и соревнования повышают интерес широкого круга пользователей к использованию компьютерных средств [4].

## 1.2. Анализ методов решения

Жанр моей игры можно определить как симулятор Аркадная головоломка. Аркадная головоломка может быть реализован следующими способами:

1. Ручной способ: в виде настольной игры. Этот способ требует подготовки физических материалов, таких как игровое поле и фигуры, а также ручного полсчета очков и управление игровым процессом. Он менее динамичен, требует значительного времени на организацию не обеспечивает зрелищности, характерной для цифровых версий.
2. Автоматизированный способ: Пользователь управляет фигурами и взаимодействует с игрой через интерфейс, вводя команды, а программа рассчитывает очки, отслеживает рекорды и управляет игровой механикой (падение фигур).
3. Автоматический способ: данные об игровых действия считываются автоматически из внешнего источника, а программа обрабатывает их без участия пользователя.

Автоматический способ не может быть использован, так как игра предполагает активное участие человека.

В нашем случае будет использован автоматизированный способ.

## 1.3. Обзор средств программирования

В рамках самых быстрых и многофункциональных решений для разработки игр, сравнение будет идти между языка Golang с движком Ebiten и C# с Unity они оба идеально подходят для разработки 2D-игр, поэтому нужно провести детальное сравнение:

Go + Ebitengine имеет преимущества в функциональности для вашего проекта. Ebitengine предлагает лёгкий и минималистичный API, идеально подходящий для аркадных головоломок, обеспечивая простую реализацию игрового цикла, управления фигурами, отрисовки (например, логотипа zetris.png) и обработки ввода. Go компилируется в нативный код, что даёт высокую производительность и компактный исполняемый файл (~17 КБ), а также кроссплатформенность без сложной настройки. Ebitengine поддерживает базовые функции (графика, аудио, текст через text/v2), достаточные для курсового проекта.

Unity на C#, хотя и мощнее для сложных 2D/3D-игр, избыточен для курсового проекта: его визуальный редактор и богатый функционал (физика, анимации, UI) увеличивают сложность и размер проекта. Unity требует больше ресурсов для изучения и настройки, а итоговый файл больше (обычно мегабайты). Go + Ebitengine проще в освоении, быстрее в разработке для лёгких игр и лучше подходит для курсового благодаря минимализму, скорости и отсутствию лишних зависимостей.

Таким образом, средством программирования данного курсового проекта был выбран язык Golang и 2D движок Ebitengine.

## 1.4. Описание языка Golang

Golang — это современный язык программирования, разработанный в 2009 году компанией Google. Он ориентирован на простоту, производительность и масштабируемость. Golang обладает строгой типизацией, сборкой мусора и встроенной поддержкой конкурентности через горутины и каналы. Его синтаксис минималистичен, что делает код читаемым и легким для изучения. Go идеально подходит для разработки высокопроизводительных, кроссплатформенных приложений. Основные преимущества: кроссплатформенность, быстрая компиляция и активное сообщество.

Таблица 1 - Ключевые слова Golang

|  |  |  |  |
| --- | --- | --- | --- |
| break | case | chan | const |
| continue | default | defer | else |
| fallthrough | for | func | go |
| goto | if | import | interface |
| map | package | range | return |
| select | struct | swiftch | type |
| var | void | catch | typeof |

Таблица 2 - Основные алгоритмические конструкции Golang

|  |  |
| --- | --- |
| Оператор присваивания | package main  func main() {  x := 10 // Короткое объявление и присваивание  y = x + 5 // Присваивание существующей переменной  println(x, y) // Вывод: 10 15  } |

|  |  |
| --- | --- |
| Условный оператор | *if (условие) {(действие) } else {(альтернатива)} ;*  package main  func checkValue(x int) string {  if x > 0 {  return "Положительное"  } else {  return "Отрицательное"  }  return "Ноль"  } |
| Оператор множественного ветвления | *Конструкция switch/case аналогична конструкции if/else, так как позволяет обработать сразу несколько условий:*  package main  func getDay(day int) string {  switch day {  case 1:  return "Понедельник"  case 2:  return "Вторник"  default:  return "Другой день"  }  *После ключевого слова switch в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями, помещенными после оператора сase. И если совпадение будет найдено, то будет выполняться определенный блок сase.*  *В конце блока сase ставится оператор break, чтобы избежать выполнения других блоков.*  *Если мы хотим также обработать ситуацию, когда совпадения не будет найдено, то можно добавить блок default,.* |
| Арифметический цикл  **(**применяется, когда известно количество повторений цикла) | for *([инициализация счетчика]; [условие]; [изменение счетчика])*  func sumNumbers(n int) int {  sum := 0  for i := 1; i <= n; i++ {  sum += i  }  return sum // Сумма от 1 до n |
| Цикл с предусловием  **(**применяется, когда неизвестно количество повторений цикла) | *while (условие) { тело цикла };*  Этот цикл будет выполняться до тех пор, пока истинно *условие* (логическое выражение, возвращающее значение типа **Boolean**). При этом если это выражение сразу равно **false**, *тело цикла* не будет выполнено ни разу. Нужно очень внимательно следить за написанием *условия* и контролем завершения цикла, так как в результате ошибки цикл **while** будет повторяться бесконечное количество раз, что приведёт к "зацикливанию" и "зависанию" программы. |
| Цикл с постусловием  **(**применяется, когда неизвестно количество повторений цикла) | *do { тело цикла } while (условие);*  Повторения сначала выполняет *тело цикла*, а затем уже проверяет выполнение *условия*. Таким образом, этот вариант цикла гарантирует, что *тело цикла* будет выполнено по крайней мере один раз. И будет выполняться до тех пор, пока *условие* не станет истинным (**true**). |

# 2. ПРАКТИЧЕСКАЯ ЧАСТЬ

## 2.1. Постановка задачи

### 2.1.1. Основания для разработки

Разработка ведётся на основании задания к курсовому проекту по профессиональному модулю ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем» МДК 01.01 «Разработка программных модулей» и утверждена Институтом среднего профессионального образования.

### 2.1.2. Назначение программы

Игровая программа «Тетрис» с графическим интерфейсом позволяет игроку соревноваться с машиной, которая случайным образом генерирует одну из 7 стандартных фигур. Цель игрока — набрать максимум очков, создавая линии из фигур, которые исчезают, принося очки, а игра заканчивается при заполнении поля.

Полный текст технического задания приведен в приложении А.

## 2.2. Проектирование приложения

На этапе проектирования были разработаны диаграмма прецедентов, диаграмма последовательностей, диаграмма классов и диаграмма активности.

### 2.2.1. Диаграмма прецедентов

Диаграммы прецедентов частично описывает use case – прецедент использования проектируемой системы, давая частичное описание частичного применения системы с точки зрения условного внешнего обозревателя. При этом описание фокусируется на том, что должна делать система по отношению к своему внешнему окружению (периферии), а не то на том, как она эта делает, то есть диаграмма прецедентов есть частичная спецификация (рисунок 1).

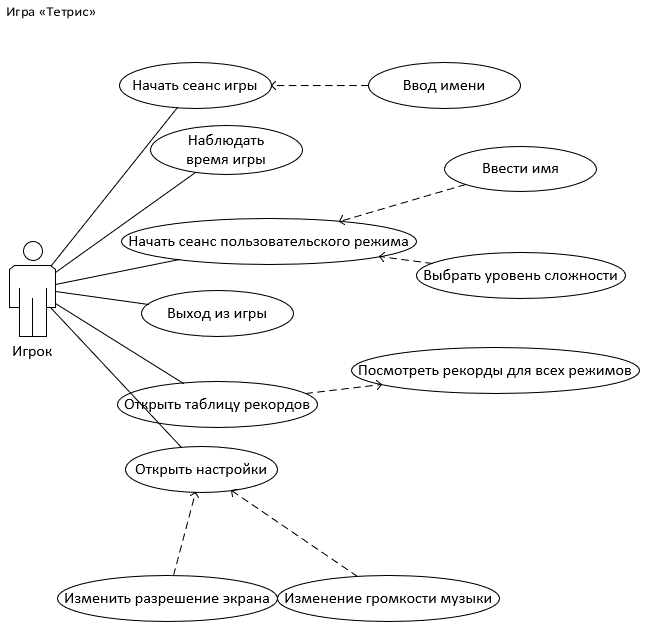


Рисунок 1 – Диаграмма прецедентов

### 2.2.2. Диаграмма последовательностей

Диаграмма последовательности отражает поток событий, происходящих в рамках варианта использования.

Все действующие лица показаны в верхней части диаграммы. Стрелки соответствуют сообщениям, передаваемым между действующим лицом и объектом или между объектами для выполнения требуемых функций. На диаграмме последовательности объект изображается в виде прямоугольника, от которого вниз проведена пунктирная вертикальная линия. Эта линия называется линией жизни (lifeline) объекта. Она представляет собой фрагмент жизненного цикла объекта в процессе взаимодействия.

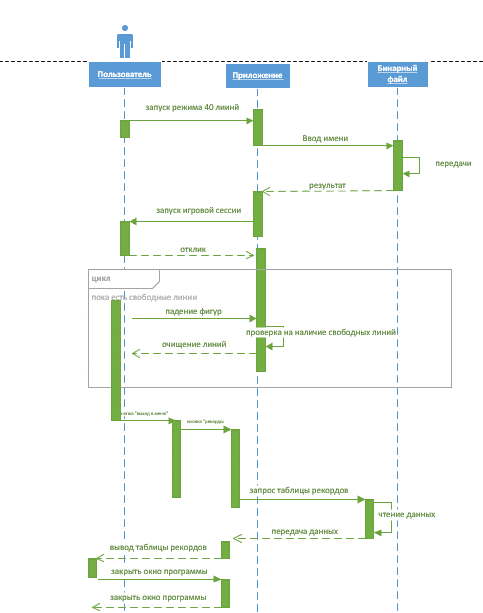


Рисунок 2 – Диаграмма последовательностей

### 2.2.3. Диаграмма классов

На рисунке 3 показана диаграмма классов проекта. Программа содержит 2 основных класса: Form1-класс формы игры, класс Login.cs содержит методы работы с бинарным файлом логинов и рекордами игроков. Этот класс использует модуль Form1 во время авторизации и при отображении таблицы рекордов.

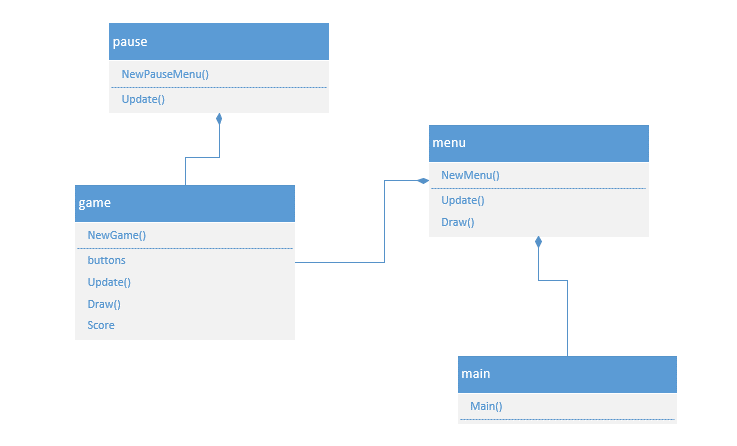


Рисунок 3 – Диаграмма классов проекта

### 2.2.4. Диаграмма активности

На рисунке 4 показана диаграмма активности (деятельности).

Диаграмма активности – это UML-диаграмма, на которой показаны действия, состояния. Под деятельностью понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов – вложенных видов деятельности и отдельных действий action, соединённых между собой потоками, которые идут от выходов одного узла ко входам другого.

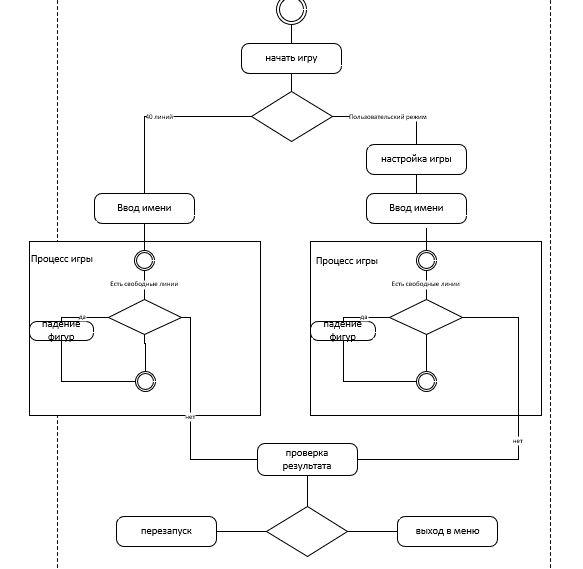


Рисунок 4 – Диаграмма активности сеанса игры

## 2.3. Текст программы

Текст программы в соответствии с ГОСТ 19.101-77 (СТ СЭВ 1626-79) и ГОСТ 19.401-79 (СТ СЭВ 3746-82) представляет собой запись программы на исходном языке программирования с необходимыми комментариями. Текст программы представляет собой документ, выполненный машинным способом, и приведен в приложении В.

## 2.4. Описание программы

### 2.4.1. Общие сведения

Игровая программа с графическим интерфейсом «ТЕТРИС»: человек играет против машины; машина генерирует каждый раз случайную из 7 стандартных фигур. Цель игрока заключается в наборе как можно большего количества очков, посредством создания линий из падающих фигур.

### 2.4.2. Функциональное назначение

Основное назначение программного продукта: компьютерная игра в жанре «аркадная головоломка».

Эксплуатационное назначение программного продукта: программа предназначена для широкого круга пользователей, без ограничения по возрасту, не требует внесения денежных средств или использования платёжных систем для игры, предназначена для развлекательных целей.

### 2.4.3. Описание логической структуры системы

Класс Game (структура в game.go) является главным модулем, отвечающим за управление игровым процессом, состояниями (меню, игра, пауза), игровым полем, фигурами и очками. Он координирует взаимодействие с другими экранами и обеспечивает основной цикл игры.

* + - 1. **Атрибуты** game.go

-grid: [gridHeight][gridWidth]string — игровое поле, хранящее позиции фигур (цветные клетки или пустые).

-score: int — текущий счёт игрока, увеличивающийся на 100 за каждую очищенную линию.

-state: GameState — текущее состояние игры (например, StateMenu, StateGame, StatePause).

* + - 1. Методы game.go

func (g \*Game) NewGame() \*Game — инициализирует игру, загружая ресурсы (шрифты, изображения, звуки) и устанавливая начальные параметры (поле, счёт, состояние).

func (g \*Game) Update() error — обновляет состояние игры: обрабатывает ввод (клавиши для перемещения фигур, паузы), перемещает фигуры, проверяет столкновения и очищает линии

func (g \*Game) Draw(screen \*ebiten.Image) — отрисовывает игровое поле, фигуры, счёт и элементы интерфейса в сине-ночной теме

func (g \*Game) Start40Lines() — запускает режим «40 линий», сбрасывая поле и генерируя первую фигуру.

func (g \*Game) StartCustomMode() — запускает пользовательский режим с настройками скорости и ограничения линий.

func (g \*Game) UpdateGameTimer() — управляет таймером падения фигур, перемещая их вниз с интервалом fallSpeed и начисляя 100 очков за очищенную линию.

func (g \*Game) LoadAssets() — загружает ресурсы при запуске игры.

func (g \*Game) Exit() — завершает игру, очищая состояние и закрывая приложение.

func (g \*Game) HandleInput() — обрабатывает ввод игрока (например, стрелки для перемещения, пробел для мгновенного падения).

func (g \*Game) ClearLines() — проверяет и удаляет заполненные линии, обновляя счёт

### 2.4.4. Используемые технические и программные средства

Для нормального функционирования данной информационной системы необходим компьютер, клавиатура, мышь и следующие технические средства:

* процессор Intel или другой совместимый;
* объем свободной оперативной памяти ~500 Кб;
* объем необходимой памяти на жестком диске ~3Мб;
* стандартный VGA-монитор или совместимый;
* стандартная клавиатура;
* манипулятор «мышь»;
* дополнительно: звуковая карты, колонки или наушники.

### 2.4.5. Вызов и загрузка

Программа может быть загружена как с диска, так и с жесткого диска. В последнем случае требуется предварительно переписать программу с диска на жесткий диск.

Исполняемым файлом программы является файл Bang.exe. Для его запуска необходимо дважды щелкнуть по исполняемому файлу левой кнопкой мышки.

## 2.5. Руководство оператора

### 2.5.1. Назначение программы

* 1. Основное назначение программного продукта заключается в организации игры с графическим интерфейсом «АНИМИРОВАННЫЙ ТЕТРИС»: человек играет против машины; машина генерирует каждый раз случайную из 7 стандартных фигур. Цель игрока заключается в наборе как можно большего количества очков, посредством создания линий из падающих фигур. После создания линии она исчезает, а игроку начисляются очки, чем больше линий очистилось за один раз, тем больше очков получает игрок. Игра заканчивается, когда у игрока заканчивается свободное место для появления новых фигур.

### 2.5.2. Выполнение программы и сообщения оператору

Для запуска программы дважды щелкните левой кнопкой мыши по исполняемому файлу Tetris.exe.

Процесс игры, этапы работы приложения показаны в виде иллюстраций в приложении Б.

При запуске приложения открывается главное меню с логотипом zetris.png и кнопками для выбора режима: «40 линий», «Пользовательский», «Рекорды», «Настройки» или «Выход». Пользователь может начать игру, выбрав режим «40 линий» или «Пользовательский» и введя имя игрока. Перед началом игры игроку присваивается начальный счёт 0 очков. В процессе игры пользователь управляет падающими фигурами, используя клавиши (стрелки для перемещения, Z/X для поворота, пробел для мгновенного падения). При заполнении горизонтальной линии она исчезает, и игроку начисляются очки: 100 очков за каждую очищенную линию. В режиме «40 линий» игра заканчивается после очистки 40 линий или при заполнении игрового поля, а в пользовательском режиме — при заполнении поля или, если выбрано ограничение, после очистки 40 линий. При завершении игры отображается таблица рекордов, показывающая максимальный счёт для режимов «40 линий» и «Пользовательский» разных игроков. Новый рекорд записывается, если текущий счёт превышает предыдущий рекорд игрока в соответствующем режиме.

### 2.6.1. Объект испытаний

Объектом испытаний является игровая программа Tetris.exe. Игровая программа «Тетрис» с графическим интерфейсом позволяет игроку соревноваться с машиной, которая случайным образом генерирует одну из 7 стандартных фигур, когда игрок собирает линию фигур, линия очищается, а игроку насчитываются очки

2.6.2. Цель испытаний

Целью испытаний является проверка соответствия программы требованиям Технического Задания.

### 2.6.3. Требования к программе

В процессе испытаний подлежат проверке следующие требования к программе:

#### 2.6.3.1. Требования к функциональным характеристикам

**1 версия:**

* программа должна генерировать фигуры в случайном порядке из перечня фигур;
* программа должна предоставлять возможность пользователю переворачивать фигуры в разные стороны;
* программа должна очищать линию при ее сборке;
* программа должна подсчитывать очки игрока по принципу:
  + 1 очищенная линия – 100 очков;
  + 2 очищенных линий – 400 очков;
  + 3 очищенных линий – 700 очков;
  + 4 очищенных линий – 1100 очков.
* программа должна отображать набранные очки в правом верхнем углу;
* программа должна увеличивать скорость падения фигур на одну секунду за каждые 5000 очков;
* программа должна предоставлять возможность поставить игру на паузу;
* поражение игрока должно возникать при заполнении всего поля не удаленными линиями фигур;
* в случае поражения игроку показываются кнопки перезапуска и выхода из игры и итоговый счёт.

**2 версия:**

* программа должна содержать главное меню с выбором режима: классический, режим 40 линий и пользовательский режим, где пользователь может выбрать свою скорость падения фигур;
* программа должна содержать раздел настроек в главном меню, где можно настроить громкость музыки, а также выбрать размер разрешения окна игры;
* проработка графического интерфейса;
* программа должна содержать спокойный фоновый эмбиент в главном меню и захватывающую музыку во время игры;
* программа должна записывать результаты игрока в локальную таблицу рекордов за сессию для каждого из режима;
* программа должна предоставлять выход в главное меню.

#### 2.6.3.2. Требования к информационной и программной совместимости

Для полноценного функционирования данной системы необходимо наличие операционной системы выше Microsoft Windows 7 или совместимой. Язык интерфейса – русский.

#### 2.6.3.3. Требования к маркировке и упаковке

Программа должна поставляться в виде проекта, исполняемого (еxе) файла.

#### 2.6.3.4. Требования к транспортировке и хранению

Программа распространяется в электронном виде. Требования к транспортировке и хранению не предъявляются.

### 2.6.4. Требования к программной документации

На испытания должны быть представлены следующие программные документы:

* техническое задание;
* текст программы;
* описание программы;
* руководство оператора;
* описание языка.

### 2.6.5. Средства и порядок испытаний

Для проведения испытаний необходимы следующие технические средства:

* процессор Intel или другой совместимый;
* объем свободной оперативной памяти ~500 Кб;
* объем необходимой памяти на жестком диске ~3Мб;
* стандартный VGA-монитор или совместимый;
* стандартная клавиатура;
* манипулятор «мышь»;
* дополнительно: звуковая карты, колонки или наушники.

Для проведения испытаний необходимы следующие программные средства:

Операционная система Windows 10.

Испытания проводятся в следующем порядке:

1) проверяется наличие и комплектность программной документации (п.2.6.4);

2) проверяется соответствие требованиям к маркировке и упаковке (п.2.6.3.3);

3) проверяется соответствие требованиям к функциональным характеристикам (п.2.6.3.1);

4) проверяется соответствие требованиям к информационной и программной совместимости (п.2.6.3.2).

### 2.6.6. Методы испытаний

#### 2.6.6.1. Для проверки способности программы запускать сеанс игры в режиме 40 линий необходимо:

* запустить программу;
* нажать кнопку «40 линий»;
* ввести имя в соответствующем окне;
* убедиться, что скрылись поля ввода имени и начался сеанс игры;

#### 2.6.6.2. Для проверки способности программы запускать сеанс в пользовательском режиме, необходимо:

* запустить программу;
* выбрать и нажать на кнопку «Пользовательский режим»
* выбрать наличие ограничения линий и скорость падания;
* нажать кнопку «Начать»
* ввести имя в соответствующем окне;
* убедиться, что скрылись поля ввода имени и начался сеанс игры;

#### 2.6.6.3. Для проверки способности программы сохранять лучший результат в таблице рекордов:

* запустить программу;
* нажать кнопку «40 линий»;
* ввести имя в соответствующем окне;
* убедиться, что скрылись поля ввода имени и начался сеанс игры;
* поставить игру на паузу, клавишой Esc;
* выйти в главное меню
* нажать на кнопку «рекорды»
* убедиться, что сохранен результат игры у игрока;

#### 2.6.6.4. Для проверки способности программы изменять разрешение экрана в настройках необходимо:

* запустить программу;
* нажать на кнопку «настройки»;
* выбрать другое разрешение экрана и нажать «применить»;
* убедиться, что разрешение экрана изменилось;

#### 2.6.6.5. Для проверки способности программы обеспечивать функцию безопасного закрытия:

* запустить программу;
* нажать на кнопку «выход»;
* убедиться, что программа завершила свою работу;

## 2.7. Протокол испытаний

Результаты испытаний программы представлены в таблице 1, рисунки приведены в приложении Б.

Таблица 3 – Протокол испытаний

| **№** | **Проверяемые требования** | **Сообщения программы и вводимые значения** | **Ожидаемые результаты** | **Результат** |
| --- | --- | --- | --- | --- |
| 1 | способность программы запускать сеанс игры в режиме 40 линий необходимо | * - запустить программу; * - нажать кнопку «40 линий»; * - ввести имя в соответствующем окне; | Окно ввода имени скрылось, игровой процесс запущен | Пройден  Рис. Б.1 |
| 2 | способность программы запускать сеанс в пользовательском режиме | * - запустить программу; * - нажать кнопку «Пользовательский режим»; * - ввести имя в соответствующем окне | Окно ввода имени скрылось, игровой процесс запущен | Пройден  Рис. Б.2-Б.3 |
| 3 | способность программы сохранять лучший результат в таблице рекордов | * -запустить программу; * нажать кнопку «40 линий»; * -ввести имя в соответствующем окне; * -поставить игру на паузу, клавишой Esc; * -выйти в главное меню * -нажать на кнопку «рекорды» | Результат игры сохранился под именем, введенным игроком ранее | Пройден  Рис. Б.4 – Б.6 |
| 4 | Способность программы изменять разрешение экрана | * -запустить программу; * -нажать на кнопку «настройки»; * -выбрать другое разрешение экрана и нажать «применить»; | Разрешение экрана изменено | Пройден  Рис. Б.7 – Б.8 |
| 5 | способность программы обеспечивать функцию безопасного закрытия | * -запустить программу; * -нажать на кнопку «выход»; | Игра безопасно прекратила свою работу | Пройден  Рис. Б.9 |

# ЗАКЛЮЧЕНИЕ

Разработанная в ходе выполнения курсового проекта программа удовлетворяет всем требованиям технического задания, что подтверждается протоколом испытаний.

Разработанная программа может быть использована в развлекательных целях, для повышения общей реакции и концентрации.

.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Описание языка Golang [Электронный ресурс] — URL: https://go.dev/doc/
2. Жанры игр и их особенности [Электронный ресурс] — URL: [https://sreda.temadnya.com/ 1043184354218215685/zhanry-igr-i-ih-osobennosti/](https://sreda.temadnya.com/%201043184354218215685/zhanry-igr-i-ih-osobennosti/) (дата обращения: 12.09.2020)
3. Документация 2D движка Ebitengine [Электронный ресурс] — URL: https://editengine.org
4. Стандарты Единой Системы Программной Документации:

ГОСТ 19.105-78 Общие требования к программным документам

ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом

ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению

ГОСТ 19.301-78 Программа и методика испытаний. Требования к содержанию и оформлению

ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению

ГОСТ 19.402-78 Описание программы. Требования к содержанию и оформлению

ГОСТ 19.505-79 Руководство оператора. Требования к содержанию и оформлению

ГОСТ 19.701-90 Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения

ПРИЛОЖЕНИЕ А

Министерство ОБРАЗОВАНИЯ И науки Российской Федерации  
Федеральное автономное образовательное учреждение высшего профессионального образования  
«Санкт-Петербургский государственный политехнический университет Петра Великого»  
(ФГАОУ ВО «СПбПУ)  
**Институт среднего профессионального образования**

УТВЕРЖДАЮ  
Председатель ПЦК   
\_\_\_\_\_\_\_\_\_\_\_ Сынкова А.Д.  
\_\_ .\_\_. 2025

Игра с графическим интерфейсом «Анимированный тетрис»  
**Техническое задание**  
Листов 6

ПРОВЕРИЛ  
Преподаватель   
\_\_\_\_\_\_\_\_\_\_\_ Девятко Н.С.  
01.02.2025

ВЫПОЛНИЛ  
Студент группы 32919/2   
\_\_\_\_\_\_\_\_\_\_\_ ФИО  
01.02.2025

1. ВВЕДЕНИЕ
   1. Полное наименование программной разработки: «ИГРА С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ «Анимированный тетрис».
   2. Игровая программа с графическим интерфейсом «Тетрис»: человек играет против машины; машина генерирует каждый раз случайную из 7 стандартных фигур. Цель игрока заключается в наборе как можно большего количества очков, посредством создания линий из падающих фигур. После создания линии она исчезает, а игроку начисляются очки, чем больше линий очистилось за один раз, тем больше очков получает игрок. Игра заканчивается, когда у игрока заканчивается свободное место для появления новых фигур.
   3. В соответствии с заданием программный продукт состоит из теоретической и практической частей. Теоретическая часть включает описание предметной области задачи, анализ методов её решения, обзор и обоснование выбора средств программирования. Практическая часть включает подробное описание проектирования программного продукта, разработку и реализацию на языке программирования Golang с использованием игрового 2D движка Ebitengine в среде программирования Visual Studio Code программных модулей программного продукта.
2. ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ
   1. Разработка ведётся на основании задания к курсовому проекту по профессиональному модулю ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем» МДК 01.01 «Разработка программных модулей» и утверждена Институтом среднего профессионального образования.
3. НАЗНАЧЕНИЕ РАЗРАБОТКИ
   1. Основное назначение программного продукта: компьютерная игра в жанре «тетрис».
   2. Эксплуатационное назначение программного продукта: программа предназначена для широкого круга пользователей, без ограничения по возрасту, не требует внесения денежных средств или использования платёжных систем для игры, предназначена для развлекательных целей, для тренировки внимания и скорости реакции.

4. ТРЕБОВАНИЯ К РАЗРАБОТКЕ

4.1. Требования к функциональным характеристикам:

**1 версия:**

* программа должна генерировать фигуры в случайном порядке из перечня фигур;
* программа должна предоставлять возможность пользователю переворачивать фигуры в разные стороны;
* программа должна очищать линию при ее сборке;
* программа должна подсчитывать очки игрока по принципу:
  + 1 очищенная линия – 100 очков;
  + 2 очищенных линий – 400 очков;
  + 3 очищенных линий – 700 очков;
  + 4 очищенных линий – 1100 очков.
* программа должна отображать набранные очки в правом верхнем углу;
* программа должна увеличивать скорость падения фигур на одну секунду за каждые 5000 очков;
* программа должна предоставлять возможность поставить игру на паузу;
* поражение игрока должно возникать при заполнении всего поля не удаленными линиями фигур;
* в случае поражения игроку показываются кнопки перезапуска и выхода из игры и итоговый счёт.

**2 версия:**

* программа должна содержать главное меню с выбором режима: классический, режим 40 линий и пользовательский режим, где пользователь может выбрать свою скорость падения фигур;
* программа должна содержать раздел настроек в главном меню, где можно настроить громкость музыки, а также выбрать размер разрешения окна игры;
* проработка графического интерфейса;
* программа должна содержать спокойный фоновый эмбиент в главном меню и захватывающую музыку во время игры;
* программа должна записывать результаты игрока в локальную таблицу рекордов за сессию для каждого из режима;
* программа должна предоставлять выход в главное меню.

4.2. Требования к надежности:

* использование лицензированного программного обеспечения;
* проверка программы на наличие вирусов;
* организация бесперебойного питания.

4.3. Требования к составу и параметрам технических средств

Для нормального функционирования приложения необходим компьютер, клавиатура и следующие технические средства:

- процессор Intel или другой совместимый;

- объем свободной оперативной памяти ~10 Мб;

- объем необходимой памяти на жестком диске ~50Мб;

- стандартный VGA-монитор или совместимый;

- стандартная клавиатура;

- дополнительно: звуковая карта, колонки или наушники.

4.4. Требования к информационной и программной совместимости

Для полноценного функционирования данной системы необходимо наличие операционной системы выше Microsoft Windows XP или совместимой. Язык интерфейса – русский.

4.5. Требования к маркировке и упаковке

Программа должна поставляться в виде проекта, исполняемого (еxе) файла, установщика и документации.

4.6. Требования к транспортировке и хранению

Программа распространяется в электронном виде. Требования к транспортировке и хранению не предъявляются.

4.7. Специальные требования

Теоретическая часть включает описание предметной области задачи, анализ методов её решения, обзор и обоснование выбора средств программирования. Практическая часть включает подробное описание проектирования программного продукта, разработку и реализацию на языке программирования Golang с использованием игрового 2D движка Ebitengine в среде программирования Visual Studio Code программных модулей программного продукта.

5. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

5.1. Предварительный состав программной документации:

- «Техническое задание»;

- разрабатываемые программные модули должны быть самодокументированы, т.е. тексты программ должны содержать все необходимые комментарии;

- разрабатываемое программное обеспечение должно включать справочную систему.

5.2. Перечень материалов пояснительной записки



Рисунок 1 – Перечень материалов пояснительной записки

6. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

Технико-экономические показатели не рассчитываются

7. СТАДИИ И ЭТАПЫ РАЗРАБОТКИ

Таблица 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Содержание стадии** | **Содержание этапа** | **Срок 2025 г.** | | **Форма  отчетности** | |
| **начало** | **конец** | |  |
| Техническое задание | Составление технического задания | 20.01 | 01.02 | | Техническое задание |
| Эскизный проект | Проектирование программы | 03.02 | 15.02 | | UML-диаграммы |
| Рабочий проект | Разработка спецификаций | 17.02 | 01.03 | | Спецификации отдельных компонентов |
| Составление программы | 03.03 | 29.03 | | Программная документация |
| Приёмо-сдаточные испытания | 31.03 | 12.04 | | Протокол испытаний (п. 2.7 пояснительной записки) |
| Приёмка | Защита курсового проекта | 14.04 | 19.04 | | Оценка за курсовой проект |

8. ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ

8.1. Порядок контроля

Контроль выполнения должен осуществляться руководителем курсового проекта (преподавателем) в соответствие с п.7.

8.2. Порядок приемки

Приемка должна осуществляться с участием руководителя после проведения приемо-сдаточных испытаний. В результате защиты курсового проекта должна быть выставлена оценка за курсовой проект.

ПРИЛОЖЕНИЕ Б

**Выполнение программы**

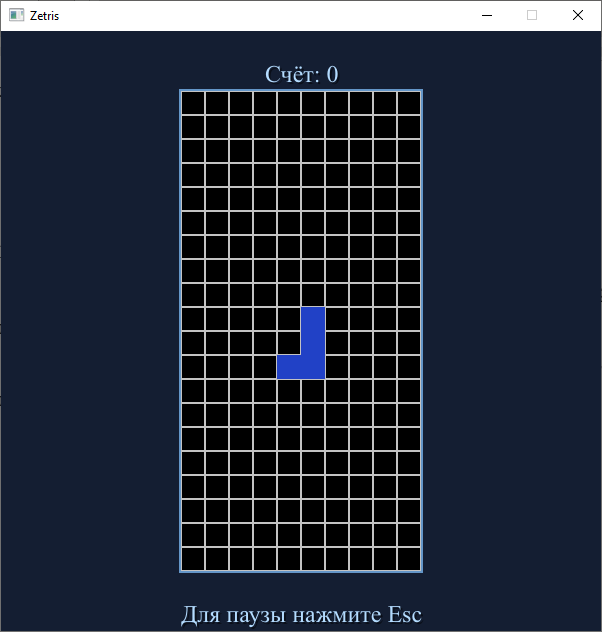


Рисунок Б.1 – игра в режиме «40 линий»

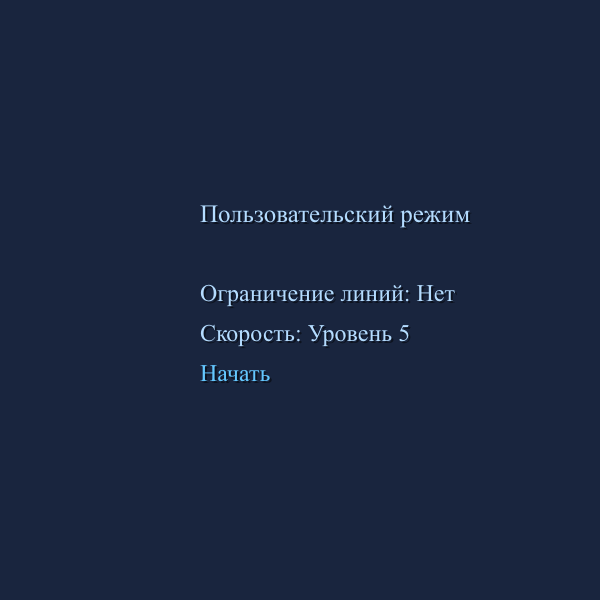


Рисунок Б.2 – настройка пользовательского режима

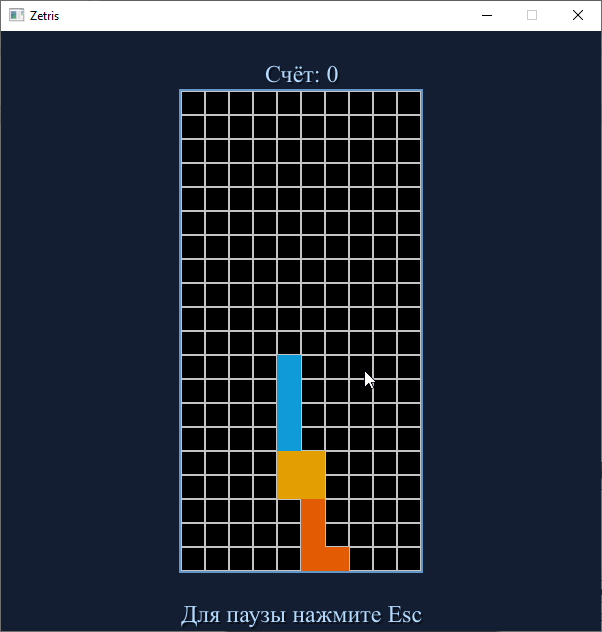


Рисунок Б.3 – пользовательский режим

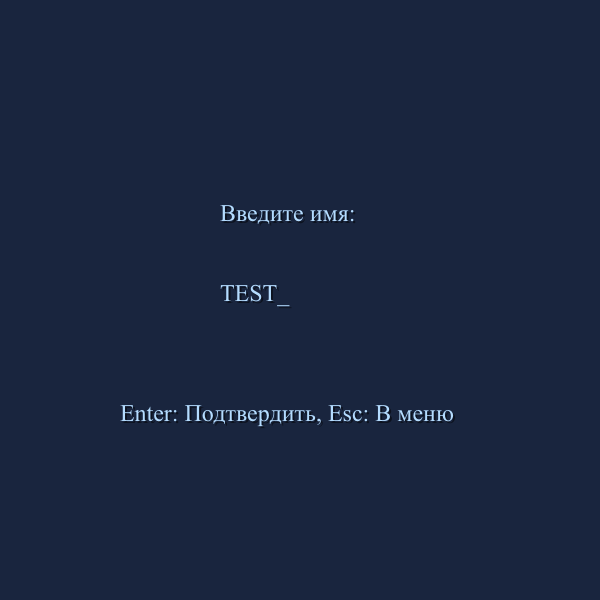


Рисунок Б.4 – ввод имени

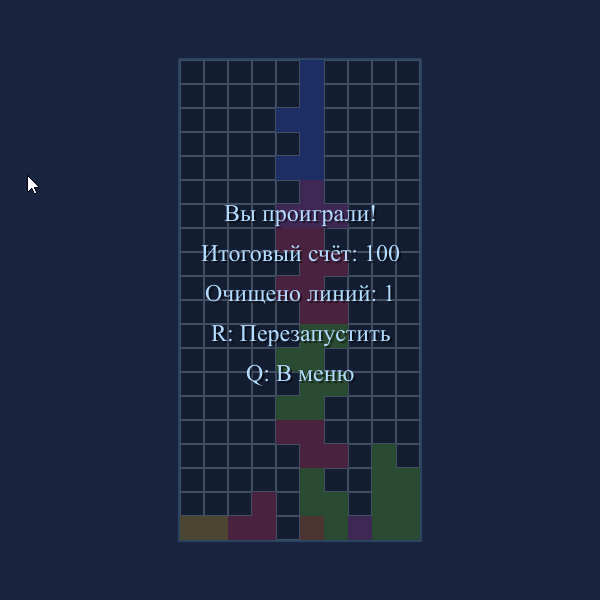


Рисунок Б.5 – запись результата

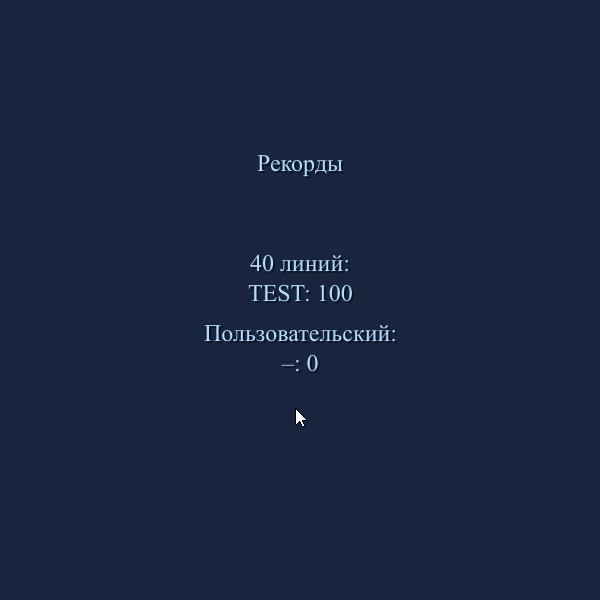


Рисунок Б.6 – лучший результат

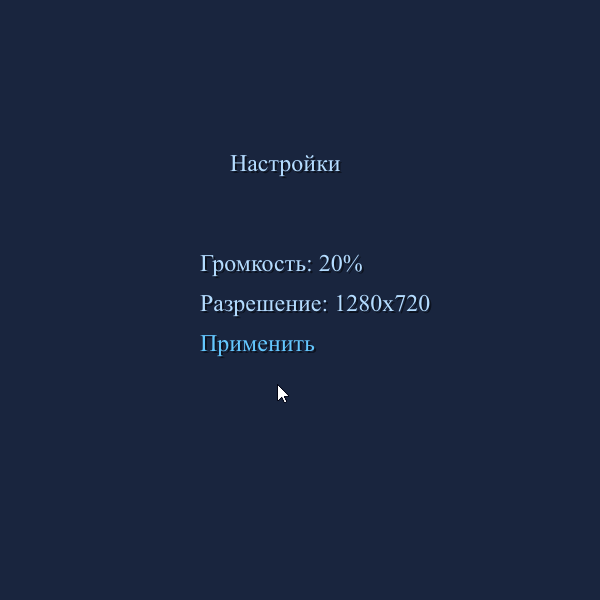


Рисунок Б.7 – панель настроек

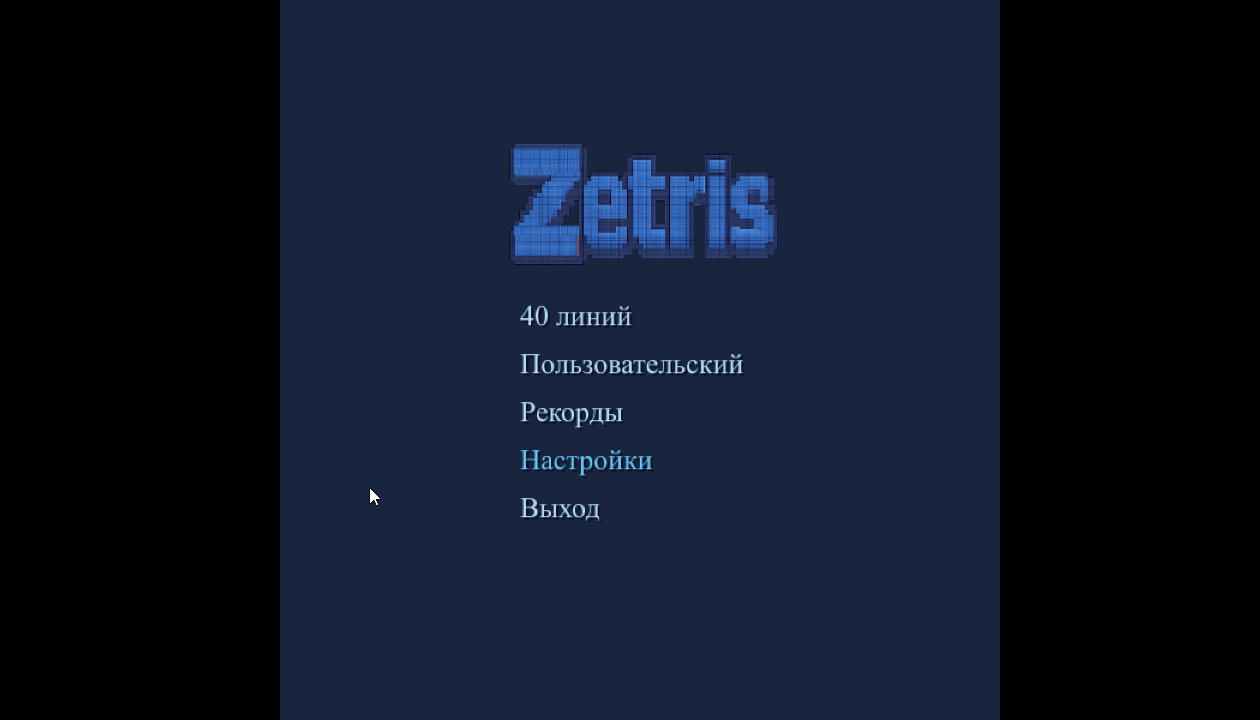


Рисунок Б.8 – Измененное разрешение



Рисунок Б.9 – Выход из игры

ПРИЛОЖЕНИЕ В

**Текст программы**

Main.go

package main

import (

"github.com/Xu3is/Zetris/src"

"github.com/hajimehoshi/ebiten/v2"

"log"

"math/rand"

"time"

)

func main() {

rand.Seed(time.Now().UnixNano())

game, err := src.NewGame()

if err != nil {

log.Fatal(err)

}

ebiten.SetWindowSize(src.ScreenWidth, src.ScreenHeight)

ebiten.SetWindowTitle("Zetris")

if err := ebiten.RunGame(game); err != nil {

log.Fatal(err)

}

}

Menu.go  
package src

import (

"github.com/hajimehoshi/ebiten/v2"

"github.com/hajimehoshi/ebiten/v2/ebitenutil"

"github.com/hajimehoshi/ebiten/v2/inpututil"

"image/color"

"log"

"os"

)

// GameState определяет текущее состояние игры

type GameState int

const (

StateMenu GameState = iota

StateGame

StatePause

StateCustomMode

StateSettings

StateEnterName

StateHighScore

)

// Menu представляет главное меню игры

type Menu struct {

game \*Game

buttons []string

selectedIndex int

logoImage \*ebiten.Image

}

// NewMenu создает новое главное меню

func NewMenu(game \*Game) \*Menu {

m := &Menu{

game: game,

buttons: []string{"40 линий", "Пользовательский", "Рекорды", "Настройки", "Выход"},

selectedIndex: 0,

}

// Загрузка изображения логотипа

path := "src/assets/images/zetris.png"

if \_, err := os.Stat(path); err == nil {

img, \_, err := ebitenutil.NewImageFromFile(path)

if err != nil {

log.Printf("Не удалось загрузить zetris.png: %v", err)

} else {

m.logoImage = img

}

} else {

log.Printf("Файл zetris.png не найден, используется запасной вариант")

m.logoImage = ebiten.NewImage(200, -100)

m.logoImage.Fill(color.RGBA{180, 220, 255, 255})

}

return m

}

// Update обновляет состояние главного меню

func (m \*Menu) Update() error {

if inpututil.IsKeyJustPressed(ebiten.KeyArrowUp) {

m.selectedIndex--

if m.selectedIndex < 0 {

m.selectedIndex = len(m.buttons) - 1

}

}

if inpututil.IsKeyJustPressed(ebiten.KeyArrowDown) {

m.selectedIndex++

if m.selectedIndex >= len(m.buttons) {

m.selectedIndex = 0

}

}

if inpututil.IsKeyJustPressed(ebiten.KeyEnter) {

switch m.buttons[m.selectedIndex] {

case "40 линий":

m.game.start40Lines()

case "Пользовательский":

m.game.state = StateCustomMode

case "Рекорды":

m.game.state = StateHighScore

case "Настройки":

m.game.state = StateSettings

case "Выход":

os.Exit(0)

}

}

return nil

}

// Draw отрисовывает главное меню

func (m \*Menu) Draw(screen \*ebiten.Image) {

overlay := ebiten.NewImage(ScreenWidth, ScreenHeight)

overlay.Fill(color.RGBA{20, 30, 50, 192})

screen.DrawImage(overlay, &ebiten.DrawImageOptions{})

// Отрисовка логотипа

if m.logoImage != nil {

op := &ebiten.DrawImageOptions{}

logoWidth, \_ := m.logoImage.Size()

scale := 250.0 / float64(logoWidth) // Масштаб 250 пикселей

op.GeoM.Scale(scale, scale)

op.GeoM.Translate(float64(ScreenWidth/2-125), float64(ScreenHeight/2-250)) // Поднимаем ещё выше

screen.DrawImage(m.logoImage, op)

}

for i, button := range m.buttons {

y := ScreenHeight/2 - 50 + i\*40

var clr color.Color = color.RGBA{180, 220, 255, 255}

if i == m.selectedIndex {

clr = color.RGBA{100, 200, 255, 255}

}

if m.game.font != nil {

drawText(screen, button, ScreenWidth/2-100, y, clr, m.game.font, i == m.selectedIndex)

}

}

}

Game.go   
package src

import (

"fmt"

"github.com/hajimehoshi/ebiten/v2"

"github.com/hajimehoshi/ebiten/v2/audio"

"github.com/hajimehoshi/ebiten/v2/inpututil"

"github.com/hajimehoshi/ebiten/v2/text/v2"

"image/color"

"time"

)

const (

ScreenWidth = 600

ScreenHeight = 600

gridWidth = 10

gridHeight = 20

cellSize = 24

keyRepeatDelay = 150 \* time.Millisecond

keyRepeatInterval = 50 \* time.Millisecond

lockDelayDefault = 500 \* time.Millisecond

lockDelayLimit = 5 \* time.Second

)

type SpeedLevel struct {

fallSpeed float64

level int

}

type HighScore struct {

Name string

Score int

}

var speedLevels = []SpeedLevel{

{fallSpeed: 0.1, level: 1},

{fallSpeed: 0.0667, level: 2},

{fallSpeed: 0.05, level: 3},

{fallSpeed: 0.04, level: 4},

{fallSpeed: 0.0333, level: 5},

}

type Game struct {

settingsMenu \*SettingsMenu

grid [gridHeight][gridWidth]string

currentPiece \*Piece

nextPiece \*Piece

score int

fallTime float64

fallSpeed float64

isPaused bool

isGameOver bool

images map[string]\*ebiten.Image

font \*text.GoTextFace

lastUpdate time.Time

keyLastAction map[ebiten.Key]time.Time

keyPressStart map[ebiten.Key]time.Time

lockDelayStart time.Time

isPieceGrounded bool

state GameState

lastState GameState

menu \*Menu

pauseMenu \*PauseMenu

customMode \*CustomMode

enterName \*EnterNameScreen

highScoreScreen \*HighScoreScreen

isLimitedTo40Lines bool

isCustomSpeed bool

clearedLines int

menuPlayer \*audio.Player

customPlayer \*audio.Player

gamePlayer \*audio.Player

classicPlayerName string

customPlayerName string

classicNameEntered bool

customNameEntered bool

classicHighScore HighScore

customHighScore HighScore

}

func NewGame() (\*Game, error) {

g := &Game{

fallSpeed: speedLevels[0].fallSpeed,

images: make(map[string]\*ebiten.Image),

lastUpdate: time.Now(),

keyLastAction: make(map[ebiten.Key]time.Time),

keyPressStart: make(map[ebiten.Key]time.Time),

state: StateMenu,

lastState: StateMenu,

classicPlayerName: "",

customPlayerName: "",

classicNameEntered: false,

customNameEntered: false,

classicHighScore: HighScore{Name: "", Score: 0},

customHighScore: HighScore{Name: "", Score: 0},

}

g.settingsMenu = NewSettingsMenu(g)

err := g.loadAssets()

if err != nil {

return nil, err

}

g.menu = NewMenu(g)

g.pauseMenu = NewPauseMenu(g)

g.customMode = NewCustomMode(g)

g.enterName = NewEnterNameScreen(g)

g.highScoreScreen = NewHighScoreScreen(g)

g.currentPiece = g.newPiece()

g.nextPiece = g.newPiece()

return g, nil

}

func (g \*Game) Update() error {

// Управление музыкой при смене состояния

if g.state != g.lastState {

if g.menuPlayer != nil && g.menuPlayer.IsPlaying() {

g.menuPlayer.Pause()

g.menuPlayer.Rewind()

}

if g.gamePlayer != nil && g.gamePlayer.IsPlaying() {

g.gamePlayer.Pause()

g.gamePlayer.Rewind()

}

if g.customPlayer != nil && g.customPlayer.IsPlaying() {

g.customPlayer.Pause()

g.customPlayer.Rewind()

}

switch g.state {

case StateMenu:

if g.menuPlayer != nil {

g.menuPlayer.Rewind()

g.menuPlayer.Play()

}

case StateGame:

if g.isCustomSpeed {

if g.customPlayer != nil {

g.customPlayer.Rewind()

g.customPlayer.Play()

}

} else {

if g.gamePlayer != nil {

g.gamePlayer.Rewind()

g.gamePlayer.Play()

}

}

case StateCustomMode, StatePause, StateSettings, StateEnterName, StateHighScore:

// Музыка не играет

}

}

if g.state == StateEnterName {

err := g.enterName.Update()

if err != nil {

return err

}

return nil

}

if g.state == StateHighScore {

err := g.highScoreScreen.Update()

if err != nil {

return err

}

return nil

}

if g.state == StateSettings {

err := g.settingsMenu.Update()

if err != nil {

return err

}

return nil

}

if g.state == StateMenu {

err := g.menu.Update()

if err != nil {

return err

}

return nil

}

if g.state == StatePause {

err := g.pauseMenu.Update()

if err != nil {

return err

}

return nil

}

if g.state == StateCustomMode {

err := g.customMode.Update()

if err != nil {

return err

}

return nil

}

if g.isGameOver {

if g.isLimitedTo40Lines && g.score > g.classicHighScore.Score {

g.classicHighScore = HighScore{Name: g.classicPlayerName, Score: g.score}

} else if g.isCustomSpeed && g.score > g.customHighScore.Score {

g.customHighScore = HighScore{Name: g.customPlayerName, Score: g.score}

}

if inpututil.IsKeyJustPressed(ebiten.KeyR) {

g.grid = [gridHeight][gridWidth]string{}

g.currentPiece = g.newPiece()

g.nextPiece = g.newPiece()

g.score = 0

g.fallSpeed = speedLevels[0].fallSpeed

g.isGameOver = false

g.isPaused = false

g.isPieceGrounded = false

g.keyLastAction = make(map[ebiten.Key]time.Time)

g.keyPressStart = make(map[ebiten.Key]time.Time)

g.clearedLines = 0

g.state = StateGame

}

if inpututil.IsKeyJustPressed(ebiten.KeyQ) {

g.state = StateMenu

g.grid = [gridHeight][gridWidth]string{}

g.currentPiece = g.newPiece()

g.nextPiece = g.newPiece()

g.score = 0

g.fallSpeed = speedLevels[0].fallSpeed

g.isGameOver = false

g.isPaused = false

g.isPieceGrounded = false

g.keyLastAction = make(map[ebiten.Key]time.Time)

g.keyPressStart = make(map[ebiten.Key]time.Time)

g.clearedLines = 0

g.isCustomSpeed = false

g.isLimitedTo40Lines = false

}

return nil

}

if inpututil.IsKeyJustPressed(ebiten.KeyEscape) {

if !g.isPaused {

g.isPaused = true

g.state = StatePause

} else {

g.state = StateMenu

g.grid = [gridHeight][gridWidth]string{}

g.currentPiece = g.newPiece()

g.nextPiece = g.newPiece()

g.score = 0

g.fallSpeed = speedLevels[0].fallSpeed

g.isGameOver = false

g.isPaused = false

g.isPieceGrounded = false

g.keyLastAction = make(map[ebiten.Key]time.Time)

g.keyPressStart = make(map[ebiten.Key]time.Time)

g.clearedLines = 0

g.isCustomSpeed = false

g.isLimitedTo40Lines = false

}

}

if g.isPaused {

return nil

}

if inpututil.IsKeyJustPressed(ebiten.KeyLeft) {

g.movePiece(-1, 0)

}

if inpututil.IsKeyJustPressed(ebiten.KeyRight) {

g.movePiece(1, 0)

}

if inpututil.IsKeyJustPressed(ebiten.KeyDown) {

g.movePiece(0, 1)

}

keys := []ebiten.Key{ebiten.KeyLeft, ebiten.KeyRight, ebiten.KeyDown}

for \_, key := range keys {

if ebiten.IsKeyPressed(key) {

now := time.Now()

if \_, exists := g.keyPressStart[key]; !exists {

g.keyPressStart[key] = now

}

if now.Sub(g.keyPressStart[key]) >= keyRepeatDelay {

lastAction, exists := g.keyLastAction[key]

if !exists || now.Sub(lastAction) >= keyRepeatInterval {

switch key {

case ebiten.KeyLeft:

g.movePiece(-1, 0)

case ebiten.KeyRight:

g.movePiece(1, 0)

case ebiten.KeyDown:

g.movePiece(0, 1)

}

g.keyLastAction[key] = now

}

}

} else {

delete(g.keyLastAction, key)

delete(g.keyPressStart, key)

}

}

if inpututil.IsKeyJustPressed(ebiten.KeyZ) {

g.rotatePieceCounterClockwise()

}

if inpututil.IsKeyJustPressed(ebiten.KeyX) {

g.rotatePiece()

}

if inpututil.IsKeyJustPressed(ebiten.KeySpace) {

for g.movePiece(0, 1) {

}

g.fixPiece()

g.clearLines()

g.currentPiece = g.nextPiece

g.nextPiece = g.newPiece()

g.isPieceGrounded = false

if g.checkCollision(g.currentPiece) {

g.isGameOver = true

}

g.lastUpdate = time.Now()

}

if time.Since(g.lastUpdate).Seconds() >= g.fallSpeed {

if !g.movePiece(0, 1) {

if !g.isPieceGrounded {

g.isPieceGrounded = true

g.lockDelayStart = time.Now()

}

} else {

g.isPieceGrounded = false

}

g.lastUpdate = time.Now()

}

if g.isPieceGrounded {

now := time.Now()

isMoving := ebiten.IsKeyPressed(ebiten.KeyLeft) || ebiten.IsKeyPressed(ebiten.KeyRight) || ebiten.IsKeyPressed(ebiten.KeyDown)

lockDelay := lockDelayDefault

if isMoving && now.Sub(g.lockDelayStart) < lockDelayLimit {

lockDelay = lockDelayLimit

}

if now.Sub(g.lockDelayStart) >= lockDelay {

g.fixPiece()

g.clearLines()

g.currentPiece = g.nextPiece

g.nextPiece = g.newPiece()

g.isPieceGrounded = false

if g.checkCollision(g.currentPiece) {

g.isGameOver = true

}

}

}

if !g.isCustomSpeed {

scoreThreshold := g.score / 5000

if scoreThreshold >= len(speedLevels) {

scoreThreshold = len(speedLevels) - 1

}

g.fallSpeed = speedLevels[scoreThreshold].fallSpeed

}

g.lastState = g.state

return nil

}

func (g \*Game) clearLines() {

linesCleared := 0

for i := gridHeight - 1; i >= 0; i-- {

filled := true

for j := 0; j < gridWidth; j++ {

if g.grid[i][j] == "" {

filled = false

break

}

}

if filled {

linesCleared++

for j := i; j > 0; j-- {

g.grid[j] = g.grid[j-1]

}

g.grid[0] = [gridWidth]string{}

i++

}

}

g.score += linesCleared \* 100

g.clearedLines += linesCleared

if g.isLimitedTo40Lines && g.clearedLines >= 40 {

g.isGameOver = true

}

}

func (g \*Game) Draw(screen \*ebiten.Image) {

// Установка сине-ночного фона

screen.Fill(color.RGBA{20, 30, 50, 255})

if g.state == StateSettings {

g.settingsMenu.Draw(screen)

return

}

if g.state == StateMenu {

g.menu.Draw(screen)

return

}

if g.state == StatePause {

g.pauseMenu.Draw(screen)

return

}

if g.state == StateCustomMode {

g.customMode.Draw(screen)

return

}

if g.state == StateEnterName {

g.enterName.Draw(screen)

return

}

if g.state == StateHighScore {

g.highScoreScreen.Draw(screen)

return

}

offsetX := (ScreenWidth - gridWidth\*cellSize) / 2

offsetY := (ScreenHeight - gridHeight\*cellSize) / 2

// Отрисовка рамки вокруг игрового поля

border := ebiten.NewImage(gridWidth\*cellSize+4, gridHeight\*cellSize+4)

border.Fill(color.RGBA{100, 150, 200, 255}) // Светло-голубая рамка

borderOp := &ebiten.DrawImageOptions{}

borderOp.GeoM.Translate(float64(offsetX-2), float64(offsetY-2))

screen.DrawImage(border, borderOp)

// Отрисовка игрового поля

for i := 0; i < gridHeight; i++ {

for j := 0; j < gridWidth; j++ {

op := &ebiten.DrawImageOptions{}

op.GeoM.Translate(float64(j\*cellSize+offsetX), float64(i\*cellSize+offsetY))

if g.grid[i][j] != "" {

screen.DrawImage(g.images[g.grid[i][j]], op)

} else {

screen.DrawImage(g.images["boardcell"], op)

}

}

}

// Отрисовка текущей фигуры

for i, row := range g.currentPiece.shape {

for j, cell := range row {

if cell != 0 {

op := &ebiten.DrawImageOptions{}

op.GeoM.Translate(float64((g.currentPiece.x+j)\*cellSize+offsetX), float64((g.currentPiece.y+i)\*cellSize+offsetY))

screen.DrawImage(g.currentPiece.image, op)

}

}

}

if g.isGameOver {

overlay := ebiten.NewImage(ScreenWidth, ScreenHeight)

overlay.Fill(color.RGBA{20, 30, 50, 192})

op := &ebiten.DrawImageOptions{}

screen.DrawImage(overlay, op)

if g.isLimitedTo40Lines && g.clearedLines >= 40 {

if g.font != nil {

// Центрирование текста

winText := "Вы выиграли!"

w, \_ := text.Measure(winText, g.font, 24)

drawText(screen, winText, ScreenWidth/2-int(w/2), ScreenHeight/2-100, color.RGBA{180, 220, 255, 255}, g.font, false)

}

} else {

if g.font != nil {

// Центрирование текста

loseText := "Вы проиграли!"

w, \_ := text.Measure(loseText, g.font, 24)

drawText(screen, loseText, ScreenWidth/2-int(w/2), ScreenHeight/2-100, color.RGBA{180, 220, 255, 255}, g.font, false)

}

}

if g.font != nil {

// Центрирование текста

scoreText := fmt.Sprintf("Итоговый счёт: %d", g.score)

w, \_ := text.Measure(scoreText, g.font, 24)

drawText(screen, scoreText, ScreenWidth/2-int(w/2), ScreenHeight/2-60, color.RGBA{180, 220, 255, 255}, g.font, false)

linesText := fmt.Sprintf("Очищено линий: %d", g.clearedLines)

w, \_ = text.Measure(linesText, g.font, 24)

drawText(screen, linesText, ScreenWidth/2-int(w/2), ScreenHeight/2-20, color.RGBA{180, 220, 255, 255}, g.font, false)

restartText := "R: Перезапустить"

w, \_ = text.Measure(restartText, g.font, 24)

drawText(screen, restartText, ScreenWidth/2-int(w/2), ScreenHeight/2+20, color.RGBA{180, 220, 255, 255}, g.font, false)

menuText := "Q: В меню"

w, \_ = text.Measure(menuText, g.font, 24)

drawText(screen, menuText, ScreenWidth/2-int(w/2), ScreenHeight/2+60, color.RGBA{180, 220, 255, 255}, g.font, false)

}

} else if !g.isPaused {

if g.font != nil {

// Центрирование текста "Счёт"

scoreText := fmt.Sprintf("Счёт: %d", g.score)

w, \_ := text.Measure(scoreText, g.font, 24)

drawText(screen, scoreText, ScreenWidth/2-int(w/2), 30, color.RGBA{180, 220, 255, 255}, g.font, false)

// Центрирование текста "Для паузы"

pauseText := "Для паузы нажмите Esc"

w, \_ = text.Measure(pauseText, g.font, 24)

drawText(screen, pauseText, ScreenWidth/2-int(w/2), ScreenHeight-30, color.RGBA{180, 220, 255, 255}, g.font, false)

}

}

}

func (g \*Game) Layout(outsideWidth, outsideHeight int) (int, int) {

return ScreenWidth, ScreenHeight

}

func (g \*Game) start40Lines() {

if !g.classicNameEntered {

g.state = StateEnterName

g.enterName.nextState = StateGame

g.enterName.isCustomMode = false

return

}

g.grid = [gridHeight][gridWidth]string{}

g.currentPiece = g.newPiece()

g.nextPiece = g.newPiece()

g.score = 0

g.fallSpeed = speedLevels[0].fallSpeed

g.isGameOver = false

g.isPaused = false

g.isPieceGrounded = false

g.keyLastAction = make(map[ebiten.Key]time.Time)

g.keyPressStart = make(map[ebiten.Key]time.Time)

g.isLimitedTo40Lines = true

g.isCustomSpeed = false

g.clearedLines = 0

g.state = StateGame

}