# Диплом на тему

# «Реализация алгоритма динамической сложности»

# Введение

В современном мире игровая индустрия становится все более динамичной и разнообразной, привлекая к себе внимание миллионов игроков по всему миру. Разработка игр является сложным и творческим процессом, который требует не только хорошего владения техническими навыками, но и умения создавать увлекательный геймплей. Игровая индустрия и ее методы постепенно проникают во многие другие отрасли, такие как образование (развивающие игры), наука и инженерия (моделирование и симуляция), медицина (виртуальная реальность для обучения хирургии и реабилитации), психология (использование игр для диагностики и лечения различных психических расстройств), бизнес (симуляторы управления и тренинги по развитию навыков руководителя) и даже архитектура (виртуальные прогулки по будущим строениям). Все больше отраслей начинают использовать игровые технологии для улучшения процессов обучения, тренировок, моделирования и развлечений, что подтверждает значительное влияние игровой индустрии на современный мир. В своей работе я обращаю внимание на взаимодействие игры с игроком, т.е. на сложность ее прохождения.

Традиционно видеоигры имеют один или несколько статических уровней сложности, из которых игрок может выбирать. Статическая сложность может вызвать у игрока раздражение, если игра слишком сложная, или скуку, если игра слишком легкая. В то время как некоторые разработчики игр могут решить, что установленная статическая сложность необходима для поддержания атмосферы игры, другие могут приоритизировать удовольствие игрока.

В данной работе я представляю алгоритм и его реализацию для нового типа системы динамической корректировки сложности, которая может быть реализована без необходимости большого объема тестирования для балансировки, полагаясь вместо этого на понимание Разработчиком игры ее особенностей и способов прохождения.

Существуют различные подходы к реализации динамической регулировки сложности:

1. Алгоритмический подход к динамической регулировке сложности в играх основан на наборе правил или критериев, которые определяют изменение уровня сложности в зависимости от поведения игрока, его успехов или неудач. Этот подход позволяет автоматически адаптировать игровой процесс к уровню навыков и предпочтениям игрока, обеспечивая оптимальное игровое взаимодействие.

2. Подход с использованием искусственного интеллекта (ИИ) в динамической регулировке сложности в играх предполагает использование алгоритмов ИИ для анализа поведения игрока, прогнозирования его действий и эффективного регулирования сложности игры в реальном времени. Использование ИИ позволяет создать более интеллектуальную систему адаптации сложности, которая может учитывать более широкий спектр факторов и обеспечивать более персонализированный игровой опыт.

3. Подход с использованием машинного обучения в динамической регулировке сложности в играх предполагает обучение модели на основе данных о поведении игроков, чтобы предсказывать оптимальный уровень сложности и автоматически адаптировать его в процессе игры. Этот подход позволяет создавать более гибкие и индивидуализированные системы регулировки сложности, способные быстро реагировать на изменения в поведении игроков и обеспечивать оптимальный баланс между вызовом и удовлетворением от игры.

Алгоритмический подход к реализации динамической регулировки сложности имеет несколько преимуществ перед подходом с использованием искусственного интеллекта (AI) и машинного обучения:

1. Простота и понятность: Алгоритмический подход обычно более прост в понимании и реализации по сравнению с AI и машинным обучением. В алгоритмическом подходе разработчики могут создать правила и логику регулировки сложности, основываясь на знаниях и опыте. Это может быть более предсказуемым и управляемым, особенно для меньших проектов или тех, где появляются ограничения на вычислительные ресурсы.

2. Гибкость: Алгоритмический подход позволяет разработчикам более гибко настраивать параметры и правила регулировки сложности. Они могут легко внести изменения в алгоритмы и адаптировать их под нужды конкретной игры или ситуации. Это удобно для быстрого экспериментирования и обновления игры.

3. Отсутствие зависимости от больших объемов данных: В алгоритмическом подходе нет необходимости в большом объеме данных для обучения модели AI или машинного обучения. Это может быть выгодно для разработки проектов с ограниченными ресурсами или ограниченным доступом к данным.

4. Быстрая и легкая адаптация: Алгоритмический подход позволяет быстро адаптироваться к изменяющимся условиям и требованиям. Поскольку алгоритмы обычно выполняются непосредственно в игре или программе, нет необходимости в сложной интеграции или переобучении модели AI при изменении условий или новых требованиях.

5. Прозрачность и контроль: В алгоритмическом подходе легче отслеживать и контролировать, какие правила и логика применяются для регулировки сложности. Разработчики имеют полный контроль над алгоритмами и могут точно настроить их поведение в соответствии с конкретными потребностями и желаниями пользователей.

Однако, стоит учитывать, что AI и машинное обучение могут вносить некоторые преимущества в реализацию динамической регулировки сложности, особенно когда имеется доступ к большому объему данных и ситуациях, где правила и логика более сложны. Подход с использованием AI может быть более адаптивным и способным к обучению на основе реального поведения игроков. Такой подход может использоваться в более сложных или прогрессивных проектах, где данные и сообщество игроков хорошо подходят для обучения модели.

# Теоретическая часть.

## Цель работы

Целью моей работы является создание алгоритма динамической сложности и разработка игры, на основе которой я смогу его применить. В качестве инструментов разработки я выбрал игровой движок unity и язык программирования C#, т.к. хорошо познакомился с ними во время курса обучения.

## Средства

1. Unity — это движок, который позволяет создавать приложения и игры. Он является одним из наиболее популярных кроссплатформенных редакторов кода с необходимым инструментарием. Через него можно писать софт для ПК, мобильных устройств и приставок.

2. C# — это современный и универсальный объектно-ориентированный язык программирования, разработанный компанией Microsoft. C# широко используется для создания приложений на платформе .NET и в различных областях разработки, включая игровую индустрию. Язык C# обладает простым синтаксисом, поддерживает множество современных функций программирования и имеет богатую стандартную библиотеку классов.

## Основные понятия

* **DDA –** Алгоритм динамической сложности (dynamic difficulty algorithm).
* **Параметры игры** - простые значения, такие как, например, здоровье игрока или скорость движения или более глобальные значения, такие как как внешние действующие силы, воздействие среды и т.п. (GV)
* **Поведение игры** – то, как ведет себя игра, для усложнения или облегчения игрового процесса. Другими словами, как среда будет препятствовать игроку для выполнения его задач. (GB)
* **Стиль игры –** некоторые игры предлагают различные условия победы, из которых игрок может выбрать, но даже в играх с одним условием победы игроки могут играть по-разному, чтобы завершить свою задачу. Например, в шутере игрок может выбрать подойти ближе к врагу, чтобы максимизировать урон, наносимый врагу, за счет получения большего урона самому (прозвище "Танк"), или он может выбрать стрелять с расстояния, нанося меньший урон в секунду, но существенно уменьшая риск получения урона (пассивный стиль игры).
* **Навык игрока –** абстрактный термин, описывающий способность игрока выполнить поставленные задачи.

# Практическая часть

# Алгоритм

1. У игры будет свое **состояние сложности**, характеризующие сложность игры и изменяемость параметров. Определятся оно будет простым уравнением. Будет задано начальное состояние сложности **DD**(default difficulty) и множители, изменяющие исходное значение переменной для изменения сложности **MV**(multiplier variable). Таким образом сложность **D**(difficulty) будет:

**D = MV \* DD**

1. Со стороны разработки, должны быть определены стили прохождения игры и метрики игрока. Используя метрики игрока, которые собираются во время прохождения только из последнего сегмента игры либо из всей игровой сессии, либо взвешенное среднее из двух, или какой-то другой вариант. Результат работы этих функций должен быть числом в диапазоне от 0 до 1, где 0 означает, что игрок не использует этот стиль вообще, а 1 означает, что игрок полностью использует этот стиль.

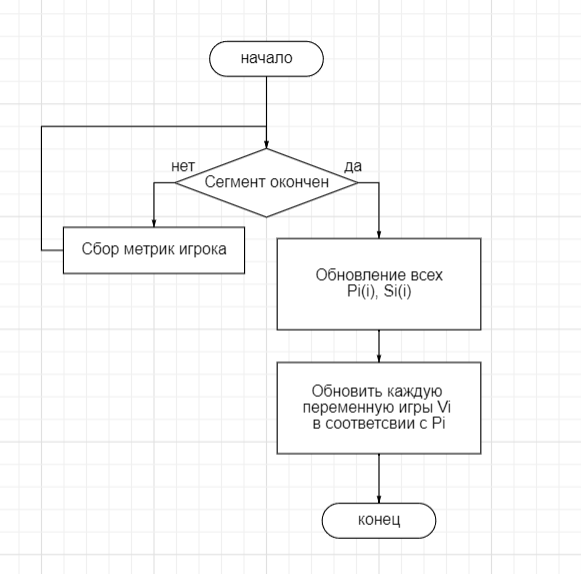
Основная цель такого подхода состоит в том, чтобы формально определить стиль игры игрока, основываясь на собранных метриках, и использовать эти значения для динамической регулировки сложности игры, адаптируя ее к предпочтениям и навыкам игрока.

1. Мы определяем **Функции навыка** – функции показывающее навыки и умение игрока. Они использует метрики игрока, определенные разработчиком. Результат работы этой функции должен быть числом в диапазоне от -1 до 1. Каждая функция будет связана с определенным стилем игры и будет отображать сбалансированность сложности. 1 означает, что игра является очень несбалансированной в пользу игрока (игрок наверняка преодолеет текущий вызов), а значение -1 означает, что игра является очень несбалансированной против игрока (игрок наверняка проиграет текущий вызов). Значение 0 означает, что игра идеально сбалансирована, и поэтому изменения сложности не требуются.
2. Также введем **матрицу противодействия**. Значение матрицы будет определять, как сильно игровой параметр или поведение игры будет «противодействовать» стилю игры. Другими словами, насколько сложнее будет игра при выбранном стиле игры, когда разница между начальным состоянием игрового параметра и его текущем состоянием будет расти. Или насколько более активным будет поведение игры. Значения матрицы (Cij) будут также в диапазоне от -1 до 1. Чем ближе к -1, значит высокие показатели параметров игры и активности поведения игры будут легче будет игроку. Если же значение матрицы ближе к 1, то наоборот, высокие параметры и активность поведения игры будут усложнять игровой процесс. 0 означает что параметры и поведение игры не влияет на сложность определённого стиля игры.
3. Обновление состояния. Для каждого стиля игры Pi и для каждого игрового параметра/активности поведения игры Vj, с соответствующем значением матрицы противодействия Cij и функции навыка Si. Множители изменения сложности MVj будут меняться : MVj = Uij.

Uij – функция U(Si, Pi, Cij) с свойствами:

* (Si = 0 или Pi = 0 или Cij = 0) ⬄ Uij = 0
* Uij пропорциональна Si Pi Cij
* Sgn(Si\*Cij)=sgn(Uij)

### Блок схема:



### Реализация

Реализовывать этот алгоритм я буду на разработанной мною игре жанра dropdown shooter - жанр видеоигр, в котором игрок управляет персонажем или транспортным средством, который стреляет во врагов, находящихся выше персонажа. В таких играх камера обычно расположена сверху, что позволяет игроку видеть всю область игрового поля. Я посчитал этот жанр достаточно простым для показательной реализации задуманного алгоритма.

Игра будет состоять из бесконечного числа уровней. В каждом уровне будет генерироваться определенное число врагов и определенное число монет. Цель игрока, либо собрать нужное число монет, либо уничтожить нужное число врагов.

Таким образом игра состоит из следящих игровых объектов:

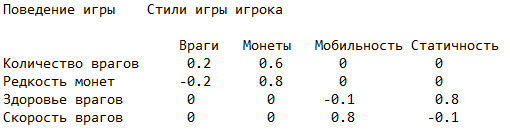
* Игрок, изначально вооружен орудием ближнего боя, но может подбирать орудия дальнего боя.
* Орудие дальнего боя, объекты, которые игрок может подобрать и использовать для более эффективной борьбы с врагами. Количество таких оружий будет случайно разбросано по уровням и их количество определяется количеством врагов
* Враги, вооружены огнестрельным оружием и обладают характеристиками скорости и здоровья, которые будут манятся. Именно эти характеристики будут отображать аспект «поведение игры» и препятствие игроку.
* Монеты, статичные объекты, случайно разбросанные по карте.

В этой реализации будут использоваться две пары взаимообратных стиля игры, для того чтобы использовать один алгоритм для каждой пары. Первый будет отвечать за количество монет и поверженных врагов для выполнения условия победы на следящем уровне, т.е. в зависимости от того, как именно способом игрок прошел уровень, будет вычисляться необходимые значения для следящего уровня. Второй будет определять силу врагов в непосредственном столкновении с игроком. Оба стиля и их взаимодействия будут описаны ниже.

Начальная сложность будет задаваться мной вручную, затем алгоритм будет подбирать данные, которые игрок будет генерировать своим прохождением и менять состояние игры.

Функции навыка будут определяться потраченным здоровьем игрока.

Матрица противодействия:



В строках слева представлено поведение игры, т.е. размещенное количество врагов и монет на уровне, здоровье и скорость врагов на уровне. Эти параметры будут меняться в зависимости от действий игрока, который в свою очередь может прибегнуть к тактикам, представленным справа сверху.

Игрок может достигать выигрыша путем победы над врагами, в ответ на это игра будет немного увеличивать количество врагов и немного добавлять количество монет на следящем уровне. Здоровье и скорость врагов не будут завесить от этой тактики.

Если же игрок захочет пройти уровень, собрав все монеты, то игра будет увеличивать количество врагов и уменьшать количество монет на уровне (повышать редкость). Так же, как и с предыдущем стилем здоровье и скорость врагов никак не будут на это реагировать.

Что касается стратегии сражения с противниками, то игрок может активно двигаться, чтобы избежать попадания, но тем самым он понижает свою точность попаданий по противнику. Поэтому игра будет увеличивать скорость врагов и слегка понижать их запас здоровья.

И обратная ситуация, когда игрок малоподвижен, чтобы больше попадать по врагам, но рискуя при этом своим здоровьем, тогда игра увеличит здоровье врагов и немного понизит их скорость.

Стратегии сражения не влияют на количество размещенных врагов и монет на карте.

Алгоритм стиля игры, основывающийся на мобильности игрока. Игрок может выбрать стратегию активного движения, либо наоборот, наименьшего движения. По сути, это два стиля игры, но т.к. они являются противоположностям, то их расчет можно произвести в одном алгоритме.

Алгоритм повышения характеристик врагов

Приверженность игрока к мобильному и стационарному стилю игры я определяю количеством поворотов за бой, если много, то мобильный, иначе стационарный. Соответственно если за время боя игрок делал много поворотов, то повышается скорость врагов, еще не вступивших с ним в бой. В обратном случае увеличивается здоровье врагов. Размер изменений определяется определять потраченным за сегмент здоровьем.

Алгоритм размещения врагов и монет на уровне.

Я ввожу два множителя на повышения и уменьшения монет и врагов на следящем уровне (т.к. на уровне собственного опыта игры во время разработки, темп на повышение и понижение должен быть разным).

Если победа достигается путем сбора монет, то на следующем раунде нужно уменьшить количество монет на карте (тут еще нужно понять какой фактор усложняет игру, редкость монет или их количество) и увеличить количество врагов. Если же наоборот победа достигается путем уничтожения всех врагов, то проверяется потраченное за раунд здоровье, если игрок теряет много здоровья, то количество врагов уменьшиться, если же нет, то немного увеличиться. Т.е. направление на изменение монет и врагов определяется стилем игры, а их количество определяется потраченным здоровьем.

Генерация каждого уровня происходит за счет генерации двумерного массива, в соответствии с которым, расставляются игровые объекты по следующему правилу:

* «#» - стена.
* «.» - проход.
* «S» - начальное положение игрока
* «M» - положение врага.
* «С» - положение монеты.
* «G» - положение оружия

// тут можно вставить код

Для генерации массива карты, я сделал отдельный класс. В нём реализуется такой алгоритм.

Сперва заполняется прямоугольник символами «#». Затем случайным образом определяется число от 4 до 11, которое определяет размеры коридора, который наносится на карту. Далее между коридорами проделываются пути, путь проделывается так, что хотя бы один проход между коридорами обязан быть. Так у нас получается карта ограниченная стеной и имеющая коридоры с проходами шириной от 4 до 11 блоков.

После постройки карты, на нее случайным образом наносятся другие игровые объекты. Но я сделал опцию, позволяющую объекту не генерироваться в указанном радиусе другого объекта. Чтобы игрок в начале уровне не появлялся в кругу врагов или чтобы монеты не скапливались друг с другом. (В дальнейшем можно поиграться с радиусом, как с еще одной характеристикой поведения игры). Изначально я планировал использовать алгоритм Эллера для генерации лабиринтов, но я не придумал как совместить его реализацию с случайным размещением игровых объектов.

Вставить

“Данный раздел стоит разделить на две части: Первая пойдет как общее описание, вторая (здесь, по моему, только матрица противодействия к ней относится) – применение к конкретной игре. Вторую часть нужно будет вынести за описание самой игры.”