Содержание

Аннотация **3**

1 Основные термины, определения и сокращения **4**

2 Постановка задачи **5**

3 Актуальность и значимость **5**

4 Предлагаемые подходы и методы **5**

4.1 Unity 5

4.2 Photon 6

5 Игровая архитектура **7**

5.1 Создание колод 7

5.2 Матч 8

5.3 Пошаговый принцип 8

5.4 Синхронизация 9

5.5 Happens-before 9

5.6 Розыгрыш карты 11

5.7 Порядок ходов 11

6 Видеоигры жанра ККИ (существующие решения) **12**

6.1 Hearthstone 12

6.2 Duelyst 2 13

7 Заключение, дальнейшее развитие **14**

Список литературы **15**

Аннотация

За последние 10 лет рынок видеоигр претерпел серьёзные изменения. В настоящее время пользователей стали больше интересовать сессионные игры (Dota 2, Counter-Strike: Global Offensive, League of Legends), нежели серьёзные одно- или многопользовательские AAA-проекты (серии игр TES, Fallout, Assasin's Creed и многие, многие другие). Конечно, видеоигры, требующие максимального погружения и отдачи от игрока никуда не делись и даже не отошли на второй план, но появилась новая ниша, проекты для которой достаточно стремительно и довольно серьёзно развиваются.

При этом можно в некотором роде рассуждать о примитивизации игрового процесса - ключевое отличие сессионных игр в том, что не нужно продумывать развитие персонажа, либо же тщательно следить за сюжетной линией. Сюжет в таких играх уходит даже не на второй - на третий план: чаще всего подаётся в виде каких-то сторонних продуктов - фильмов, мультфильмов, комиксов и т.д.

Но нельзя не отметить при этом успех игр, которые вместе с сессионностью приносят серьёзный, продуманный игровой процесс - самым ярким представителем является Dota 2 и в целом все MOBA-игры. Поэтому можно предположить, что сессионные игры, которые в то же время обладают продуманным геймплеем, который заставляет игрока достаточно глубоко погружаться, могут завоевать достаточно большую аудиторию.

Поэтому мы решили разработать компьютерную карточную игру с элементами пошаговой стратегии - такая игра бы сочетала премущества сессионной игры (гибкость по отношению к потребностям игрока - возможность тратить на неё совсем немного времени и быть т.н. казуальным игроком, или тратить гораздо больше и добиваться успехов) и классической видеоигры (глубокое погружение в геймплей).

В данной работе мы ставим перед собой задачу создать компьютерную карточную онлайн-игру. Основной целью работы в сроки курсовой работы является создание прототипа игры, затем - доработки, доведение до релиза. В конечном итоге планируем получить полноценный коммерческий проект.

Ключевые слова

Компьютерные игры, компьютерные карточные онлайн-игры, геймдев

1 Основные термины, определения и сокращения

Геймплей - игровой процесс.

Компьютерная карточная игра (ККИ) - видеоигра, основой которой является карточная игра по некоторым правилам. Ключевое отличие от обычных карточных игр заключается в том, что игрок может некоторым способом создавать новые карты и конструировать из них колоды, с которыми он играет против других игроков. Одна из самых популярных на данный момент - Heartstone.

Сессионные игры - жанр видеоигр, где весь игровой процесс сосредоточен на коротких игровых промежутках времени в среднем от 15 до 60 минут - *сессиях*. Игровой прогресс между сессиями не сохраняется. Отсюда следует, что игроки в пределах сессии обладают равными игровыми возможностями, вне зависимости от игрового опыта.

MOBA-игры - жанр сессионных командных видеоигр. В игре представлено две команды, каждая из которых имеет свою базу и главное здание. Цель игры - разрушить вражеское главное здание. Игроки управляют чаще всего каждый отдельным, своим персонажем. Самый яркий представитель и в то же время основатель жанра - серия игр Dota (Dota 2, Dota как модификация к видеоигре Warcraft III).

Unity - кроссплатформенная среда разработки компьютерных игр. Позволяет автоматизировать создание игры на таких платформах, как Windows, Linux, Android, iOS. Среда автоматически выполняет работу игрового движка, позволяя программисту сосредоточиться на написании скриптов на *C#*.

Сцена - понятие в Unity - отдельный эпизод игры, в котором игрок имеет ограниченное управление на некотором пространстве (карте) через игровой интерфейс. Сценой может быть главное меню, бой с противником, перемещение по карте и т.п.

Photon - фреймворк для *Unity*, предоставляющий возможности для создания онлайн-игр.

Ассет (asset) - набор исходных материалов для проекта на Unity: моделей, задних планов, библиотек, фреймворков и т.д.

Альфа-версия - стадия разработки проекта - внутреннего тестирования некоторой промежуточной версии продукта с целью выявления ошибок, добавления новых функциональных возможностей.

Бета-версия - стадия разработки проекта - внутреннего и/или внешнего тестирования и отладки проекта с целью выявления максимального количества ошибок в проекте и их исправления.

Happens-before - отношение между двумя событиями А и В в распределённой системе, когда событие В произошло с учётом той информации, которая была получена от события А.

Remote-Procedure Calls(RPC) - вызов процедуры, исполнение которой осуществляется в другом адресном пространстве. Описание RPC явно или неявно включает в себя сетевой протокол для обмена данными и язык сериализации объектов.

2 Постановка задачи

В данной работе я ставил перед собой цель разработать альфа-версию компьютерной карточной онлайн-игры. В альфа-версии игроку должна быть доступна игра по сети с другими игроками и возможность создавать свои колоды карт. Опциональной задачей была дальнейшая разработка - наполнение игры новым материалом (новыми картами или расами), совершенствование дизайна игры и т.п.

3 Актуальность и значимость

Как уже было сказано выше, сейчас сессионные игры набирают всё большую популярность. Например, популярная ККИ Hearthstone в 2020 году имела около 23,5 млн. активных игроков. Но помимо этого, ККИ являются не настолько развитым жанром видеоигр, как, например, шутеры. Поэтому новые разработки в этом жанре могут достаточно серьёзно усилить рынок ККИ и привлечь новую аудиторию.

4 Предлагаемые подходы и методы

4.1 Unity

На данный момент, Unity является одним из самых простых и при этом самых мощных инструментариев для разработки игр [6].

Работа с Unity сочетает в себе как работу с самим редактором, так и написание скриптов. Работа с редактором включает в себя добавление объектов на сцену, изменение интерфейса на экране, настройка правильной иерархии объектов.

Вторая часть работы с Unity - это написание скриптов. Скрипт - это файл с программным кодом, написанным на языке *C#*. Как правило, скрипт описывает поведение некоторого

объекта на карте (и в таком случае должен наследоваться от класса MonoBehaviour), но можно и описывать стандартные классы в скриптах, как в классическом ООП.

На рисунке 4.1 изображены главные составляющие интерфейса Unity: центральное окно, по умолчанию в нём можно переключаться между вкладками сцены и игрового поля, слева - иерархия объектов, справа - инспектор объекта: поле, в котором можно настраивать отдельные свойства объекта, снизу - файловый менеджер проекта

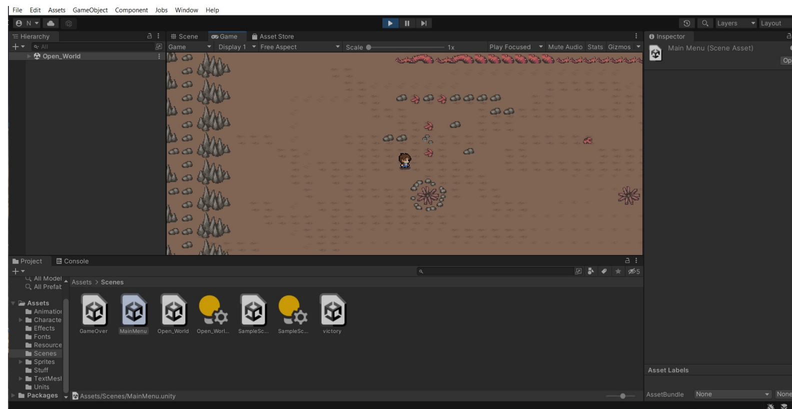

Рис. 4.1: Интерфейс Unity

В случае, если класс наследуется от `MonoBehaviour`, в нём необходимо определить две функции: `Start()` и `Update()`. Первая функция вызывается в момент создания объекта, вторая - на каждом кадре видеоигры.

На рисунке 4.2 изображены поля инспектора. С помощью них можно настраивать публичные переменные класса скрипта, привязанного к объекту, добавлять новые компоненты объекта и настраивать их.

4.2 Photon

Photon - это фреймворк, с помощью которого можно создавать онлайн-игры в Unity [5]. Кроме того, разработчики Photon также предоставляют аренду серверов для разработчиков.

Базовое понятие в мультиплеерной игре с помощью Photon - это комната. Комната в Photon - это логический контейнер, представляющий собой сессию игры. К этой сессии могут свободно подключаться игроки или покидать её.

Существует две основных версии фреймворка Photon: Photon Unity Networking 1 (PUN1) и Photon Unity Networking 2 (PUN2). PUN1 основывается на технологии RPC (Remote Procedure Call), а PUN2 предоставляет более высокоуровневое API в виде встроенных функций.

Рис. 4.2: Инспектор объекта.

При этом в проекте использовались RPC-вызовы, а не их более высокоуровневые аналоги из PUN2. Мотивация для этого будет описана ниже.

5 Игровая архитектура

Для того, чтобы играть в KARD, у игрока должны быть следующие возможности: создание колоды и поиск другого игрока для проведения матча.

5.1 Создание колод

Игрок может создавать колоды с различным набором карт. Каждая из карт принадлежит к определённой расе. В данный момент в игре 4 расы: люди, демоны, нежить и эльфы.

Для работы с картами создан абстрактный класс `Card`, который является интерфейсом для класса любой карты. Класс объявляет единственный абстрактный метод - `Play` - принимающий на вход координаты на поле (см. ниже) и реализующий эффект карты (например, призывая существо)

Для работы с колодами карт создан класс `Deck`, содержащий саму колоду карт, название колоды, расу, к которой принадлежит колода.

5.2 Матч

Матчи проходят на поле размером 15 на 7. В самом начале боя у каждой из рас на поле боя появляется генерал - особое существо. Чтобы победить, игрок должен убить вражеского генерала. Генерал каждой из рас отличается своей особенностью (например, у людей генерал имеет повышенную броню)

На рисунке 5.1 изображён пример начала матча, если оба игрока играют за расу людей.

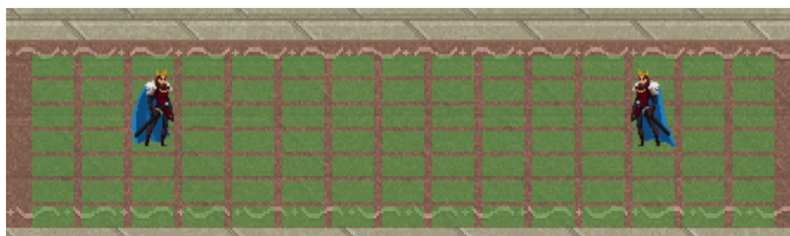

Рис. 5.1: Начало матча в KARD

5.3 Пошаговый принцип

Вся игра разделена на раунды. В момент хода своего существа, игрок может разыграть на поле карту - не более одной карты за весь раунд.

Каждое существо на поле боя обладает своими параметрами. Один из таких параметров - инициатива. Очерёдность хода существ в пределах раунда упорядочена по их инициативе.

Для реализации очерёдности хода используется класс InitiativeManager. Класс содержит две очереди, каждая из которых является экземпляром класса `SortedSet<Creature>`, где `Creature` - это класс, описывающий существо на поле и имеющий компаратор, с помощью которого существа упорядочиваются в пределах очереди. Класс имеет две очереди, в первой находятся существа, ход которых ещё не наступил в этом раунде, во второй - которые уже совершили ход. После конца хода существо перемещается из первой очереди во вторую. В конце раунда очереди меняются местами.

5.4 Синхронизация

Как уже было сказано выше, средства PUN1 и PUN2 предоставляют различные возможности для синхронизации объектов: высокоуровневые и низкоуровневые.

Примером высокоуровневой синхронизации является функция `PhotonNetwork.Instantiate()`. Эта функция после обращения к серверу создаёт объект на всех клиентах, который в течении своего времени жизни будет автоматически синхронизирован между всеми клиентами.

Данная реализация является неэффективной в случае игры KARD. Для более эффективной реализации необходимо использовать тот факт, что игра является пошаговой.

Для примера рассмотрим перемещение существа. Если данный объект - существо - было создано с помощью функции `PhotonNetwork.Instantiate()`, то все его перемещения будут синхронизироваться между игроками. В то же время, если в момент начала хода существа оба игрока обладали информацией о его положении, то единственной необходимой информацией о перемещении существа будут являться координаты её точки перемещения. Получив такую информацию, оба клиента смогут независимо обработать перемещение существа на карте и начать следующий ход.

Именно по этой причине в проекте используются RPC-вызовы, а не высокоуровневые средства. Следствием такого использования является и возникновение задач синхронизации, которые в случае использования высокоуровневых средств решаются автоматически. Ниже рассмотрены варианты решения таких задач, которые были реализованы в проекте.

5.5 Happens-before

В случае использования RPC-вызовов необходимо тщательно следить за соблюдением между событиями отношения happens-before, пример такого отношения изображён на рисунке 5.2. Нарушение таких событий может повлечь за собой аномалии в игровом процессе для наблюдателя. Аналоги таких аномалий в более классических распределённых системах описаны во многих источниках, например, [2].

К примеру, для взаимодействий на карте мы используем только RPC-вызовы. В таком случае, при нестабильной сети и переупорядочивании сообщений, может произойти следующий сценарий:

1. Игрок 1 совершает ход. RPC-вызов A обрабатывается и результаты доходят до игрока 2.

2. Игрок 2 совершает ход. RPC-вызов B обрабатывается и результаты доходят до

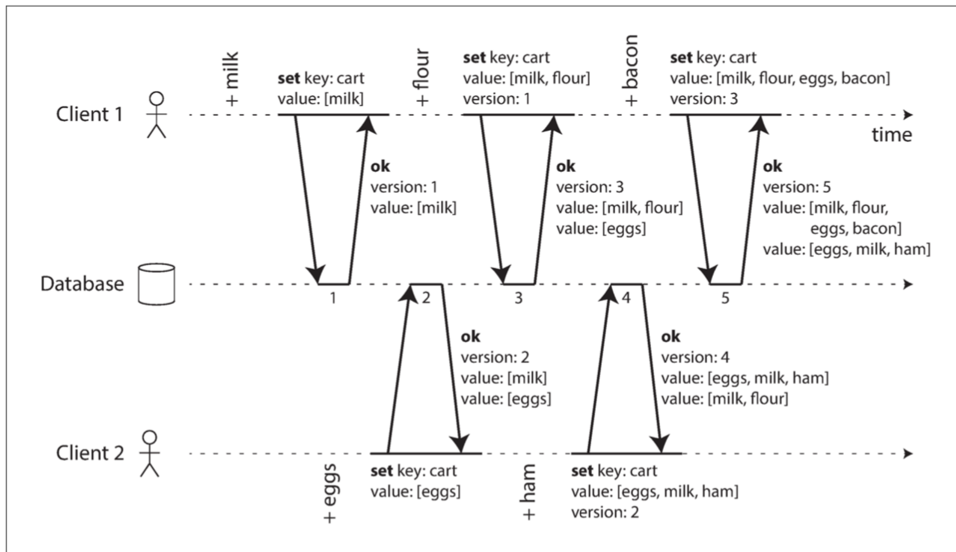

Figure 5-13. Capturing causal dependencies between two clients concurrently editing a shopping cart.

Рис. 5.2: Пример отношения happens-before

игроков 1 и 2.

3. Через некоторое время до игрока 1 доходит результат вызова A.

В этом примере наблюдается классическое нарушение принципа happens-before. Для его устранения рассмотрим, как в этом примере совершался RPC-вызов в фреймворке Photon:

```
photonView.RPC(Go, RpcTarget.All, x, y);
```

Здесь Go - названием функции, `RpcTarget.All` означает, что RPC-вызов будет отправлен всем клиентам в комнате, включая инициатора. x и y - параметры вызова (координаты на поле, куда должно отправиться существо).

В этом примере коммуникация выглядит так: Клиент 1 отправляет сообщение о вызове RPC на сервер. Сервер обрабатывает сообщение и отправляет результаты клиентам 1 и 2.

Модифицируем вызовы следующим образом:

```
Go(x, y);
photonView.RPC(Go, RpcTarget.Others, x, y);
```

Данный код предоставляет гарантию, что клиент 2 получит результат вызова RPC не раньше, чем этот вызов локально выполнится на клиенте 1. Параметр RpcTarget.Others обеспечивает вызов RPC у всех клиентов, кроме инициатора, поэтому в данном случае также меньше сообщений передаётся между клиентами и сервером.

5.6 Розыгрыш карты

Кроме того, happens-before может нарушаться не только между ходами игроков, но и в пределах хода одного игрока. Причиной тому является то, что игрок может во время своего хода также разыграть карту. При этом ход существа означает завершение хода игрока, поэтому порядок может быть только один из двух:

`Ход существа`

ИЛИ

`Розыгрыш карты` → `Ход существа`

Переупорядочивание этих событий также приведёт к аномалиям: например, у второго игрока бегущее по полю существо может начать обходить несуществующее препятствие, которое является на самом деле новопризванным существом, которого игрок пока не наблюдает.

Для решения этой проблемы написан класс `ExecutionUnit`. Данный класс обладает методом `SetAction`, который устанавливает будущее событие. Один из параметров, передающихся в этот метод - тот факт, была ли сыграна карта вторым игроком или нет. В зависимости от этого действие будет либо обработано сразу, либо будут сохранены его параметры и оно будет произведено, когда клиенту придёт RPC-вызов о том, что была разыграна карта. Во втором случае сначала будет разыграна карта на столе, и только потом существо сделает ход. Таким образом, happens-before будет сохраняться.

5.7 Порядок ходов

Подводя промежуточный итог, между игроками меняется принцип взаимодействия в сторону уменьшения синхронизации: каждый игрок явно сохраняет локальную копию очереди ходов всех существ и самих существ на поле. При получении RPC-вызова, каждый игрок производит на поле необходимые манипуляции, перемещая существ и разыгрывая карты.

Преимущество данного подхода в том, что игрок как получатель игрового опыта не будет наблюдать абсолютно никаких задержек, единственной возможной для него аномалией будет отключение второго клиента - выход оппонента из сессии и завершение игры. Причина этого заключается в том, что для игрока неразличимы сетевые задержки и долгое принятие решений оппонентом, а по описанной выше схеме действия самого игрока он наблюдает моментально, так как все такие вызовы являются локальными.

Последняя проблема, которая может возникнуть, это порядок ходов существ с одинаковой инициативой. Такое может встречаться часто, например, если два игрока играют за

одну расу - у двух генералов и по совместительству единственных существ на поле в самом начале матча одинаковая инициатива.

Наиболее честным решением проблемы в таком случае является случайное распределение всех существ в классе существ с одинаковой инициативой между собой (класс здесь употреблено в математическом смысле) Но в таком случае необходимо обеспечивать такое распределение, чтобы оно было случайным, но в то же время одинаковым для обоих игроков. Эту задачу решает класс `RandomGenerator`, генерирующий случайные последовательности в зависимости от изначального параметра `seed`. Параметр `seed` должен задаваться числом, которое заранее известно обеим клиентам (например, `currentRoom.MasterClientId` - идентификатор создателя комнаты, или любой другой параметр комнаты)

6 Видеоигры жанра ККИ (существующие решения)

6.1 Hearthstone

На данный момент одна из самых популярных ККИ в мире [4]. Пример матча в Hearthstone изображён на рисунке 6.1.

Рис. 6.1: Матч в Hearthstone.

Изначально в игре было всего 9 классов, но со временем количество расширили до 11 (см. рис. 6.2)

Также в игре есть не только режим игры против других игроков (ботов), но и арена (сбор колоды из случайных карт и игра против других до 3х поражений) и поля сражений

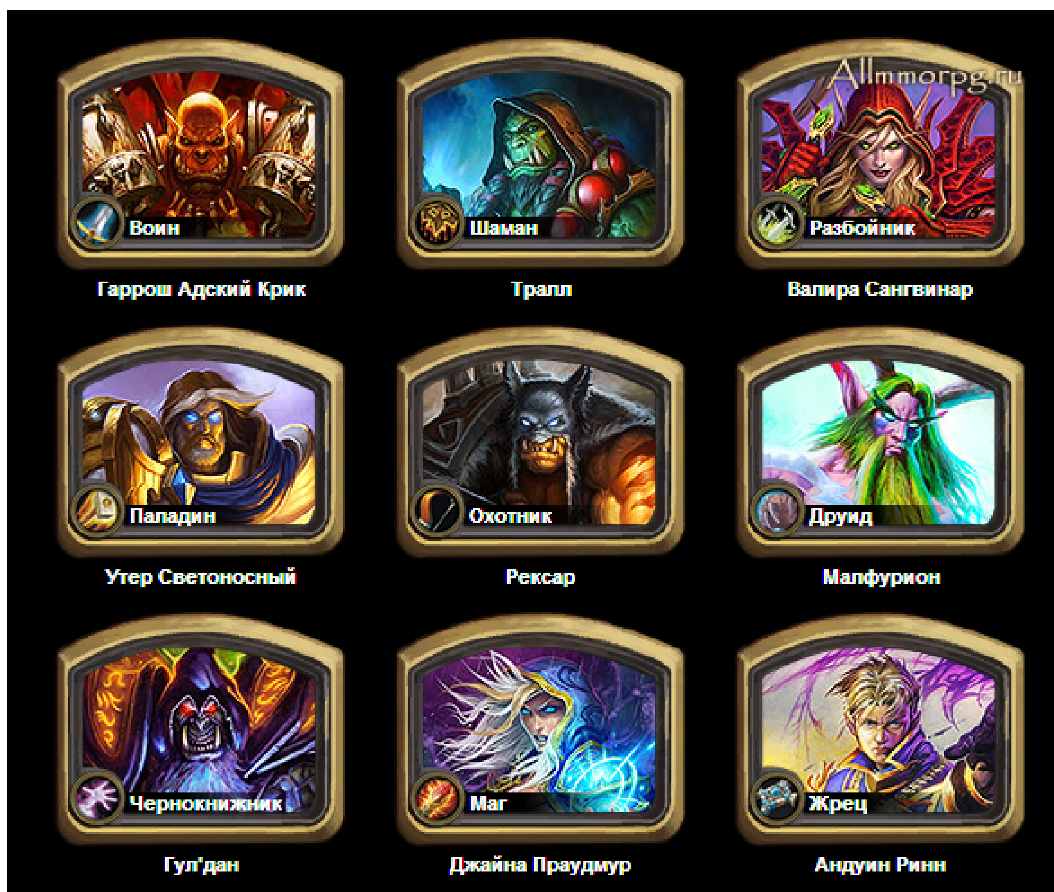

Рис. 6.2: Классы в Hearthstone.

(покупка существ за золото - аналог маны в этом режиме, и затем автобой против других игроков). В будущем, возможно, мы также расширим количество игровых режимов и для нашей игры.

6.2 Duelyst 2

Компьютерная карточная игра, обладающая схожими принципами с KARD [3]. Как выглядит матч в Duelyst 2, можно наблюдать на рисунке 6.3.

Здесь наблюдается похожий принцип - сочетания правил ККИ и пошаговой стратегии в одной игре. В KARD планируется более качественный баланс (например, сейчас в Duelyst для выравнивания преимущества первого хода на поле лежат 3 кристалла маны, которые можно собрать. Но всегда есть возможность, что один игрок соберёт все 3 кристалла, и это почти гарантированная победа), изменить сеттинг, потенциально разнообразить режимы игры. Кроме того, в Duelyst 2 каждый игрок ходит по очереди, и за ход каждый игрок использует всех существ. В KARD в игре у существ есть параметр инициативы, что делает игровой процесс более динамичным.

Рис. 6.3: Матч в Duelyst 2.

7 Заключение, дальнейшее развитие

В итоге получили альфа-версию 2D ККИ [1], которая обладает определённой глубиной геймплея.

В данный момент существует множество возможностей для дальнейшего развития. Главным образом - перевод проекта в стадию бета-тестирования: добавление новых карт, дизайнерские улучшения. Достаточно важной задачей является добавление механики получения новых карт.

Один из наиболее требующих развития и изменения компонентов проекта - это сеттинг. В настоящий момент графический компонент игры является комбинацией из множества бесплатных ассетов.

Список литературы

[1] *KARD*. URL: https://github.com/MikhailSazonov/KARD-Game.

[2] Martin Kleppmann. “Designing Data-Intensive Applications”. В: O’Reilly, 2016.

[3] *Вебсайт Duelyst 2*. URL: https://duelyst2.com.

[4] *Вебсайт Hearthstone*. URL: https://hearthstone.blizzard.com/ru-ru.

[5] *Документация по Photon*. URL: https://doc.photonengine.com/pun/current/getting-started/pun-intro.

[6] *Документация по Unity*. URL: https://docs.unity.com/.