Blade II Online

Blade II Online is an online multiplayer, 1v1 card-game based on a minigame (Blade, later Blade 2) found in some of Nihon Falcom's[1] RPG games including Trails of Cold Steel II[2] and Tokyo Xanadu eX+[3]. After each player draws a random set of cards, they must take turns placing them, trying to obtain a higher final score or to force their opponent into a position where there are no legal moves left for them to play.

My main motivation for choosing to make this game is fairly simple - after graduating, I will be applying for jobs in Japan, and my first choice is the aforementioned Nihon Falcom. I feel that it would be a great boost to my employability if I was able to present a complete, polished product that may even be commercially viable if combined with their IP.

As an aside, until now I have not had a chance to develop either a card game, or a game with online multiplayer support. And even though Nihon Falcom focuses on RPG development, I don't feel like I have the resources required in order to make a full RPG style game, especially in the typical Japanese aesthetic (I am neither a 2D nor a 3D artist), and so for me this is the most relevant game I feel comfortable attempting to produce.

Blade II Online is designed to offer fast-paced gameplay with minimal downtime. A user should be able to start a match within one minute of logging in to the client, including time spent waiting in the matchmaking queue. Games will last around five minutes each, depending on the speed at which the player's take their turns.

With a rapid play cycle and the Elo based high score system that updates after every match, Blade II Online presents an exciting gameplay experience with a tight feedback loop and easy to understand metrics regardless of player ability. Players will quickly find themselves competing against other players of similar skill levels and will be able to track their progress in real-time as they continue to play the game.

After downloading and installing the Blade II Online client, players will be required to either register a new account, or login with an existing one. Once logged in, they will be able to either join the matchmaking queue immediately, play practice games against an AI opponent, view the overall Elo rating leader boards, or inspect their own and other player's Elo rating.

The first time the player logs in, they will be given the option to take part in a practice match with a predetermined set of cards and moves, following the on-screen prompts until they game is complete. The player can repeat this as many times as they like, until they decide to move onto an unguided game versus the AI opponent, or begin playing real games by joining the matchmaking queue.

The matchmaking queue will attempt to match players with similar ratings against each other, though it will fall back to matching without restriction if the queue is low on players, or a suitable opponent is not found within a reasonable amount of time.

All players will start off with an Elo rating of 1200. Though this number is arbitrary, it is apparently in-line with other popular online games with Elo rating systems such as League of Legends (though a definitive source could not be found).

Match overview

The match itself features a single deck of thirty-six cards, of which thirty are randomly selected per match. Each player receives ten cards for their hand (cards that can be immediately used at the start of their turn) and receive an extra five to place in their deck (a reserve set of cards that is drawn from in certain circumstances).

- Once the match begins and the cards have been dealt, they must each draw a card from the deck and place it on the field.
- The player who draws the lower number card starts first and must place a card from their hand onto the field.
- Their opponent then has two options to progress:
    - Play a card with a value that allows them to surpass or match their opponents current total or,
    - Play a special effect card and observe its effect. Depending on the effect, they may need to play another card. Otherwise, their turn will end, and it becomes the other players turn.
- This process repeats until either player can no longer surpass their opponents' total value, they run out of legal moves, or run out of cards to play.
- Should the scores be equal after a player finishes their turn, the field is cleared, and both players must draw and play new cards.
- If the final card of the game results in the field being cleared, the match will result in a draw.

Card overview

All cards have a value that contributes to the players score - even cards with additional effects cards. Special effect cards that are not used for their effect (such as when played onto an empty field) will simply increase the score by their base value of one. Otherwise they will apply an effect, such as reversing (flipping over and removing the points gained) a card, doubling the players score, or swapping each player's score and current field.

Blade II Online – card descriptions				
ID	Count	Card Name	Value	Effect
0	2	Elliot's Orbal Staff	1	Un-reverse a reversed card on the players side of the field
1	3	Fie's Twin Gunswords	2	None
2	4	Alisa's Orbal Bow	3	None
3	4	Jusis' Sword	4	None
4	4	Machias' Orbal Shotgun	5	None
5	3	Gaius' Spear	6	None
6	2	Laura's Greatsword	7	None
7	6	Bolt	1	Reverse opponent's last played card
8	4	Mirror	1	Switches the field around, effectively swapping the played cards for each player, as well as their current scores
9	2	Blast	1	Removes a card from the opponent's hand (selected by the player)
10	2	Force	1	Doubles the players current score

Figure 1 - Blade II Online - playing cards

Special rules

- Effect cards cannot be played as the last card. If the player only has a single effect card remaining at the end of the turn, they automatically lose the game. If the player only has effect cards remaining (but has more than one), they may continue taking turns for as long as they have more than one effect card remaining.
- Effect cards placed on an empty field (i.e. at the start of the game after being drawn from the deck) simply add their value to the players score, and the effect is ignored.
- Should the scores be equal after a player finishes their turn, and a player has no cards left in their deck, the player must select a card from their hand to place on the field instead.

End conditions

1. A player ends their turn without successfully matching or surpassing their opponent's score.
2. All the cards in the game are exhausted in some way.
3. A player ends their turn with a single, effect card in their hand only.
4. A player disconnects from the match and their opponent decides to accept the match victory, rather than waiting for their opponent to reconnect and resume the game.

Once the game has ended, the outcome of the game (win/loss/draw), a log of the moves that occurred during the game, and the time of completion, are submitted to a central database which recalculates and updates each players Elo rating, as well as inserting the match data into a match history archive.

Players will be informed of their new Elo rating, and will then be given the option to have a look at their opponent's profile or return to the main lobby to begin the cycle again.

Requirements

Must Have	Should Have	Could Have	Won't Have
In-client account creation	Game controller compatibility	In-game messaging with pre-set text	PS4 build (lacking appropriate license)
Login with username and password	Server-side validation for player actions	Animated 3D avatars in menus and player profiles	Microtransactions
Login, settings, queue, game, post-game, leaderboard game states	SSL/TLS encryption for all network communications	Ability to reconnect to unfinished games	Mobile app for Elo leaderboards, user profile lookup etc.
In-client password reset via email and front-end webpage	Multi-platform builds (PC, mobile, consoles if available)	Add other players as friends and start games with them directly	In-game chat with unrestricted text input
Offline AI opponent	In-client Elo rating high score tables	Unranked queue with no visible Elo rating	Paid DLC
Optional tutorial vs. AI opponent	Classifications for each Elo range	Web-app Elo leaderboard	Card collecting mechanics
Multiplayer matchmaking queue	Username profanity filter	Web-app game client	Dynamic matchmaking queue that adapts to players based on recent win-percentage

JWT based authentication when connecting to the game server	UI animations	Free DLC	
Persistent data stored in remote SQL database	Customisable options such as volume and graphics quality	Optional advertisements	
Mouse & Keyboard compatibility	Option to remember username	Customizable card design	
WebSocket network communication with a central relay server built with Go	Cross-platform multiplayer	Customizable playing table design	
REST API server built with Go (auth, queries etc.)	Credits and license information section	Animated Login screen	
Customizable player avatars	Post-account creation email confirmation	Game launcher built with Electron	
Elo leaderboard with divisions that group players by percentile	Initial seeding period for new players where Elo shift is inflated	3D parallax effect for cards	
Client-side validation for player actions		Optional AR mode	
Fully featured UI, including window controls		Optional in-game voice chat	
Fully Localized for Japanese & English			
Background music and sound effects			
Match History			
Animated 3D avatars in-game			

Figure 2 – Blade II Online - MoSCoW requirements priority table

Literature Review

There are many comparable products available on the market today. Many of them are digital versions of existing card-games (such as Magic: The Gathering Arena, and Pokémon TCG Online). Some are based on existing characters and lore (such as Hearthstone) whilst others are largely original concepts (such as Shadowverse). Other notable examples include Microsoft Solitaire, which was introduced in Windows 3.0 to familiarize users with various methods of interacting with a PC using a mouse (Caldwell, 2019).

Digital trading card games have existing in various forms since the late 90's, with some of the earliest examples being Magic: The gathering, released in 1997 (Blevins, 2000) and Pokémon Trading Card Game, released in 1998 (Nintendo, 1998). Online Digital trading card games have also existed since the late 90's, with notable examples

being Sanctum (Jebens, 1998), and Chron X (Bregger, 2019), both of which were released in in 1997. Though these are all examples of games that feature card-collection mechanics, Blade II Online will feature a single, static deck of predetermined cards in order to stay true to the original game upon which it is based.

Blade II Online also draws from Riot Games' League of Legends. The Elo rating system and ranked matchmaking queue, combined with the overall layout and aesthetic of the client, provide great examples of how to deliver a satisfying gaming experience that also takes into consideration usability, providing a vast array of quality of life features such as a friends list, parties, cooperative matchmaking queues and an in-client marketplace.

Similar Products

Hearthstone

Figure 3 – Hearthstone gameplay screenshot (Blizzard Entertainment, 2019)

Hearthstone is an online CCG developed and published by Blizzard Entertainment. It was released on March 11th, 2014 (Zeriyah, 2014) for Windows and MacOS (Yin-Poole, 2014) and has since been made available on iOS and Android (Kollar, 2015). The game is built using Unity (Wilson, 2014).

Hearthstone combines existing lore, themes and content from the Warcraft series, and features card collecting and 1v1 turn based player versus player gameplay. Players must be conscious of their primary resource, mana, as different cards require different amounts of mana to play. All cards boast a rarity value, from Legendary (rarest), through to Epic, Rare, Common, and Free (most common) (IGN, 2014).

Players select a class (Warrior, Shaman, Rogue, Paladin, Hunter, Druid, Warlock, Mage, or Priest) and utilize four different types of card (minions, spells, weapons, and hero), aiming to reduce their opponent's health to zero whilst maintaining their own.

Hearthstone is 'fast paced' and 'is a game that's designed for anyone to come up, to pick up and play; to be a very accessible experience' (Brode and Chayes, 2013), and Blade II Online will attempt to echo these sentiments. Fast,

easy to learn gameplay will be the foundation that draws in newer players, providing a low barrier to entry that requires no knowledge of the game universe from which it hails (The Legend of Heroes series), and very little understanding of card games in general.

As a relatively new player to Hearthstone, and card-games in general, I very much appreciated how easy it was to get the hang of some of the basic mechanics of hearthstone, as well as how easy it is to get in and out of a game. I hope to be able to recreate this feeling with Blade II Online, offering similarly quick entry and a shallow learning curve that eases players in and rewards those who invest time into learning more advanced ways of playing, and the intricacies of the game.

Hearthstone, while presented as a 2D top-down card game, is in fact comprised of a combination of 2D and 3D assets and rendered entirely in 3D (Hearthstone, 2017, 04:02). 2D Paintings are mapped onto 3D models using Maya (Brode, 2013), and after angling models so that they appear natural in the single, top-down POV (Zwicker, 2013) are added into the game.

By presenting what would typically be a 2D game in three dimensions, Hearthstone is able to employ a variety of 3D effects without the need for complex shaders. Z-depth and layers are handled by world-space positioning, paving the way for cards with layers and inherent parallax effects, and indicators (such as arrows or special effects) that stand out from the background with their physical presence and accurate perspective effects.

In a similar fashion, Blade II Online will make heavy use of 3D assets to represent objects within the game. Cards, the arena, player avatars, and indicators such as arrows and special effects, will be comprised of either 3D meshes made in Maya, or 2D sprites with varying z-depths to create an illusion of depth or “thickness”.

League of Legends

Figure 4 – League of Legends gameplay screenshot[4]

League of Legends is an online MOBA developed and published by Riot Games. It was released on October 27th, 2009 for Windows (Paah, 2010), and later other platforms. The game makes use of a custom engine built specifically

for League of Legends development, with the core game being written in C++. The game client, a separate application that acts as a homepage/game launcher, is built around the CEF (Chromium Embedded Framework) and uses a combination of JavaScript/HTML/CSS and C++ (Lima Beans, 2009).

League of Legends was originally developed as a spiritual successor to the popular Warcraft 3 mod DotA (Defence of the Ancients). Merril (2009) stated that 'League of Legends came to be because a couple of very active DotA community members believed that the gameplay was so fun and innovative that it represented the spawning of a new genre and deserved to be its own professional game'.

Players (typically ten, with five on each team) select one of some 140 (at the time of writing) champions, along with a choice of two spells (again, from a larger collection that varies in size depending on the game type), and after creating a special configuration page of extra effects and abilities, they start the game. Once the game has started, each player controls their selected champion. While purchasing and upgrading items, defeating monsters, fighting the enemy players, and knocking down the enemy's towers, each team aims to destroy the enemy base whilst simultaneously defending their own.

Being a MOBA, League of Legends gameplay is completely different from the card-game that is Blade II Online. However, the ranked matchmaking system employed by League of Legends is a framework upon which many of the design decisions for the online multiplayer components of Blade II Online were based.

The most prominent feature worth mentioning is the Elo based ranked matchmaking system. Players can queue up to find a game and will be automatically placed into games with other plays of similar skill levels, earning or losing Elo rating (commonly referred to as MMR (Match Making Rating) or LP (League Points)) based on the outcome of the game.

The act of playing ranked games and testing one's mettle versus other players is both exhilarating and stressful. Earning and losing Elo rating, being promoted or demoted between ranks, and knowing that every game contributes to one's personal rating creates an incredibly fun, and somewhat addictive gameplay experience that I have personally enjoyed for many years. I hope to be able to capture some of this thrill and excitement, whilst also staying true to the core themes of the Legend of Heroes series from which Blade II Online draws inspiration.

Once players have taken part in ten or more ranked games, they will be awarded with an Elo value based on the outcome of their first ten games. With a goal of placing new players in the correct Elo as soon as possible, the first ten games are treated differently to other games, with the rate at which Elo is gained/lost increasing significantly. By doing so, players should find themselves close to their true Elo, allowing for a smoother transition into regular ranked games.

The mathematics behind calculating Elo rating is commonly known. It is of course used in many different games and sports in all manner of ways, and the calculations used for Blade II Online are shown in the **Implementation detail** section below.

The second feature of League of Legends ranked matchmaking is the division of players into different tiers and divisions.

A tier is simply a range of Elo ratings that are grouped by a named category. In League of legends, players are divided up into nine separate groups based on their Elo ratings (from lowest to highest) – Iron, bronze, silver, gold, platinum, diamond, master, grandmaster, and challenger.

Within each tier there are five divisions, ranging from five (lowest) to one (highest), which further separates the players. The exception to this is for master, grandmaster and challenger which, which are not separated into divisions – possibly due to the very small number of players that are placed within this tier.

Blade II Online will implement a similar system, but due to the considerably smaller scale, will only implement three tiers with no subdivisions. The names for the tiers are based on the lore of the Legend of Heroes universe;

- Junior Bracer
- Senior Bracer
- Grandmaster

Tools and Technologies

Game engine and language

The Blade II Online game and client will be an all-in-one application built using Unreal Engine 4. Where possible, game, engine and network communication logic will be written in C++ - otherwise, a combination of Blueprints and Unreal Engine 4's various visual editors will be used.

During the initial planning stage, various other engine / language combinations including Unity / C#, and Electron / JavaScript were considered. However, a decision was ultimately made to use of Unreal Engine 4. This is discussed below in more detail.

1. Performance

Unreal Engine 4 runs on, and supports development using C++, which is typically more performant than equivalent code written in C#, and considerably faster than code written in JavaScript. Furthermore, the runtime is smaller (takes less space on the user's device) and the memory requirements for are usually lower.

Though the article is not focused on comparing performance, Warren's (2019) investigation of the low-level features of C# reliably show that even well optimized C# runs slower than comparable C++ code, and with higher memory requirements to boot.

Performance was a key factor in the decision-making process, but with the expectation of creating a staggeringly beautiful, AAA quality visual masterpiece that demands a high-performance machine to run. Rather, the aim was to keep the game lightweight and fast on all modern machines, regardless of their hardware. Blade II Online features simple 3D graphics, many of which are simply sprites placed in 3D world-space. To live up to preconceived expectations for the resource requirements for such a game, the engine and language that offers the smallest, least resource hungry, most performant game possible was selected.

2. Post graduate employment opportunities

Due in part to an increase in adoption by high profile Japanese game franchises such as Street Fighter, Tekken, and Kingdom Hearts, Japanese adoption of Unreal Engine 4 is steadily increasing. More and more developers in Japan gain experience with Unreal Engine 4, which often leads to the natural adoption of the engine for other projects as time passes (Kawasaki, 2017). Combined with first class Japanese language support for official resources (Suzuki, 2014), experience with Unreal Engine 4 a valuable asset for those looking to work in the Japanese games industry.

With the need to save time and money being one of the main draws for the historically high cost of Japanese game development, developers have been abandoning their bespoke, in-house engines in favour of Unreal Engine 4 (Sweeney, 2018).

For these reasons, combined with Epic setting up a Japanese branch in 2010 (Famitsu, 2010), a lot of existing C++ talent, and 'a great number of universities and professional schools in Japan [adopting] UE4 into their curriculum'(Kawasaki, 2013), Unreal Engine 4 is more popular than ever, and being able to demonstrate proficiency will increase the employability of anyone wishing to apply for jobs in the Japanese gaming industry.

3. Personal development

During my year-long placement at Sony, I found myself writing mainly JavaScript and Python code during development. While these languages both have their merits and use-cases, they are not compatible with Unreal Engine 4 or Unity. Unity does feature UnityScript, a JavaScript like scripting language used for writing gameplay code, but this has been unsupported since Unity 2018.2 (Krogh-Jacobsen, 2018).

I already have a few years of experience using C# and consider myself to be relatively proficient. I began my journey of developing games with online tutorials that focused on Unity and C#. Since then I have had many opportunities to develop games using Unity at university and feel confident that I can implement anything that I might need with Unity's C# scripting interface.

Conversely I have had very little opportunity to develop games using Unreal Engine 4 and have had only a few chances to write C++ code since the end of the first year of university.

Taking all this into consideration, it seemed logical to me that in order to optimize the amount of learning and personal development achieved during this project, I should make use of an industry standard game engine that allows gameplay and engine code to be written in C++.

4. Features

Whilst both Unreal Engine 4 and Unity have a fairly similar set of tools, the former often has a richer, more powerful offering. Unreal Engine 4 is also open source, and the engine can be extended and modified if required.

Both engines for example support complex 2D user interfaces, but Unreal takes the cake with its powerful, modular, extensible UMG toolkit. UMG is fully implemented in C++ code, and can be extended either by inheriting from the UserWidget class, or by direct interaction with Slate widgets. UMG widgets can then be manipulated using the UMG UI designer, a visual UI authoring tool.

Unity does offer similar modular capabilities with its prefab system, but UI classes cannot be inherited from and must therefore be wrapped in a helper script that can only access the public interfaces of the UI class that it is handling. UI prefabs can be modified in their own prefab editor, but the majority of UI development is handled directly in the editor scene window.

Unreal Engine 4 can also be combined with any third-party libraries, as long as one can provide the dynamic libraries for each target platform. These can be called natively in C++ code without needing to use InteropServices (like one does with Unity, which introduces overhead). This will be useful when implementing the client-side network stack for Blade II Online, potentially removing the need to implement the WebSocket protocol from scratch.

Unity does have some advantages over Unreal Engine 4, however. Multi-threaded programming is made simple with Coroutines, and it's very easy to serialize and expose variables to the editor view, allowing references to game objects to be stored before runtime. Logging is lightweight and easy to use (as opposed to Unreal Engine 4's overly verbose UE_LOG() macro shenanigans) and getting references to game objects during runtime is very simple with various static Find x helper functions made available via the MonoBehaviour class.

Some of the Unreal Engine 4 features that will be used include:

- UMG and Slate UI tools
- Post Process Effects such as Bloom, Anti-Aliasing, and Post Process Materials
- Skeletal Mesh Animation System
- Sound Mixes, Sound Cues, and the Sound Cue Editor
- Niagara visual effects system
- Media Framework
- Artificial Intelligence with Behaviour Trees and Blackboards

The Go programming language

I decided to develop both the API and the game server using the Go programming language, due in part to the recent rise in interest in the Go programming language in Japan. During my placement year, many members of staff showed interest in the language, expressing a desire to learn about it and dip their toes into developing software (especially web/app servers) using Go.

In fact, the Go programming language is gaining popularity all over Japan. According to Bizreach[5] (2018), Go developers were being offered the highest median, and highest maximum salaries when compared with all other languages, as of 2018.

The ever-growing popularity of Go in Japan, along with the fact that I find it very pleasant and productive to work with, inspired me to spend some time learning Go, before eventually deciding to make use of it for my final year project.

Go is well suited for building web servers due to its simple, efficient multi-thread functionality (goroutines) and native multi-processor support. The Go runtime handles all the underlying logic in such a way that is completely transparent to the host OS. Context switching is very cheap (over 10 times cheaper than traditional threads) and inter-thread communication is lightweight (Hiwarale, 2018). This means that it's relatively straight forward to develop Golang servers that are capable of serving many concurrent users, even on limited hardware, as well as being very easy to scale.

Go also has an excellent standard library, offering many high-quality tools for a vast range of applications, including a full http library. Furthermore, the Golang community is driven largely by open-source projects, and at this point there are hundreds, if not thousands of high-quality, tried and tested industry standard packages available for more advanced tasks. One example of such a package is the chi router, which boasts usage in production by many large companies including Cloudflare and Heroku (*go-chi/chi: lightweight, idiomatic and composable router for building Go HTTP services*, 2019).

The Blade II Online Go components will make use of Gorilla's WebSocket package to implement WebSocket functionality, and Gorilla's Mux package for handling the HTTP REST API. Thanks to the fully featured standard library offered by Go, there is no need to make use any web server framework. Functionality will either be built from

scratch when simple or adopted from one of the many well used open-source packages such as those mentioned above.

REST API Implementation

The REST API server will handle user account management tasks such as creating, modifying, and getting information for user accounts. It will also act as an authentication server, provisioning authentication tokens that clients will use to connect to the Game server. It will make use of a local SQLite database for persistent storage, which it will interact with using GORM[6].

The structure of the server itself is based on the methodology described by Duncan (2019). Further inspiration is drawn from Johnsson (2019), who illustrates how database interactions can be obfuscated by restricting database interactions to a specific set of commands wrapped in functions.

The API will both receive and transmit messages with bodies in JSON format. This has many advantages over plaintext transmission, with the biggest benefit coming from the fact that JSON marshalling (the act of parsing a string into a JSON object with a particular format) implements a pseudo input sanitizer - incorrectly formatted JSON objects, will automatically throw errors when marshalled, as described in the official go website (Google, no date), removing the requirement for the developer to implement validation and sanitization and instead focus on handling the errors that are thrown.

Gorilla Mux will be used to route HTTP requests to the aforementioned functions facilitating client interactions with the internal database, such as obtaining the hashed password for a user or resetting a user's password.

Game server Implementation

The game server itself is the core of Blade II Online – all online games will be initialized and operated by this server, and after a game has finished the server will be responsible for reporting the outcome to persistent storage.

Gorilla WebSocket will be used to allow clients to connect to the server. This connection will persist throughout gameplay, during which clients can send encoded updates to the server, which are then validated and relayed to the opposite client.

WebSocket, which makes use of TCP for transmitting data (Sookocheff, 2019), was chosen over a UDP solution due to the reliable nature of WebSocket data transfer. Blade II Online is not a particularly time-sensitive game, at least not enough to warrant using the less reliable UDP protocol. Avoiding the need to implement handlers for UDP related issues, which are described by Kumar (2018), simplifies development and reduces the complexity of the game server. Implementing handling for out-of-order packets, retransmission of lost/damaged packets, congestion control, and error detection, will not only increase the time and expertise required to develop the game server, but also increase the load on the server itself – The Blade II Online server will have very limited resources due to a fairly restrictive student budget[7].

Clients will connect to the server using an authentication token obtained from the API server beforehand. Once connected, the initial HTTP request is upgraded to a WebSocket connection which is then wrapped in a container and placed into a queue.

The queue is polled regularly by the matchmaking engine, which observes various data values (time spent in queue, Elo rating etc.) in order to match two players together for a game.

Once two clients are paired up, the relay engine works together with the validation engine to blindly forward messages between the two clients, whilst also delivering command messages (such as game start, game end, and disconnection alerts) when necessary.

After the initial handshakes, including the relaying of a set of cards randomly generated server-side, clients will take turns sending data packets containing deltas that describe any changes to the state of the game that were made on their turn. Based on the methodologies described by Glazer and Madhav (2016, p. 177), these input packets will be sent to the server and then handled by the client proxy, which will subsequently request validation of the packet.

If the validation fails, the server will drop the connection and award a win to the opposing player, and a loss to the offender. This ensures that players cannot manipulate the game by intercepting game traffic within their own network and modifying the contents of packets before sending – an example of which is shown by Wu (2018) using mitmproxy[8].

If validation succeeds, the server will update the internal state of the game and then relay the delta to the opposite client. This will then trigger this client to begin their turn, during which they will be required to make a move, and therefore send a new set of deltas to the server, thus continuing the cycle.

This will continue until either one of the players wins the game or drops their connection. When a connection is dropped, all players (in this case just the opposing player) will be notified and asked if they would like to wait for their opponent to reconnect. If they choose to wait, the game will be moved into a standby queue that will periodically check with the matchmaking engine to see if the disconnected player has reconnected. As each player has a unique ID number, the server will be able to determine if connecting clients are assigned to an existing game, or if they are looking to join the matchmaking queue.

Once the game is over, the initial card selection, delta history, player ID's and match outcome is sent to the API server in order to update the database. The game server will make use of a user account with raised privileges that are required in order to POST to this API endpoint.

All data sent between the clients and server will sent as binary data and interpreted as ASCII on both ends.

As the game server is written in Go, and the game itself written in C++, it will not be possible to share the validation logic between the two sources. Thus, it will be required to implement the validation algorithm twice – once in both languages.

Glazer and Madhav (2016, p. 171) state that 'One of the cornerstones of the client-server model with an authoritative server is that the code that executes on the server is different from the code that executes on each client'. They then go on to discuss how common objects (present both client and server side) can be derived from in order to produce two classes; one for client use and one for server use.

As the Blade II Online game server is written in a different language to the game itself, special care must be taken to ensure that the common data structures (cards, and update deltas) are implemented identically in both programs, and that the code that interacts with them shares identical functionality. With this in mind, card data will be stored as an array of unsigned byte arrays, with the first dimension representing the card locations (field, player 1 deck, player 2 deck etc.), and the second dimension representing the actual cards present at that location. The data structure will be wrapped in a container class, with functions that hide the underlying implementation such as `GetPlayerDeck(player n)` and `Serialize()`.

Update deltas will be represented by a staring value (2D array indices of the card being moved, and a destination value (2D array indices of the new location of the card that was moved). Like the cards data structure, this will also be wrapped in a container class.

By ensuring the data structures are uniform, packing and unpacking of data will be implemented similarly on both ends, meaning that one implementation can act as a basis for the other. In this fashion, once the validation system has been written for one platform, it should be trivial to port it to the other due to the similar underlying data structures.

Back end infrastructure stack

Blade II Online's back-end infrastructure will rely on multiple different technologies in order to operate. Both the API and game server will be proxied by a local Nginx[9] server, all of which will run on a CentOS VPS[10]. These will then be connected to the wider internet through Cloudflare[11].

As mentioned earlier, Nginx web server will be used as a reverse proxy for the API and game servers. It's quick and simple to set up and is more than performant enough for the small scale at which Blade II Online will operate.

Using Nginx as a reverse proxy, and therefore as a barrier between the internal web server and the outer network, makes it simple to implement features such as SSL/TLS and, rate limiting. A declarative scripting language is used to write configuration scripts, which exist as physical files on the native OS' file system.

SSL/TLS support can be implemented in just a few lines of script, tweaked to block access to older, insecure protocol versions, and made valid with a self-signed SSL certificate (Ellingwood, 2017).

Similarly, rate limiting can be applied to whole servers, or specific endpoints, in just a few lines or script. This provides protection against large bursts of traffic, or attackers that seek to disrupt the hosted service by repeatedly making requests to the server (Rawdat, 2017).

The aforementioned technologies (Go servers and Nginx) will be hosted on a CentOS VPS, hosted by a third-party company. By hosting these services on an external, always-on device, Blade II Online can ensure high uptime and availability for testing and demoing the game, without the need to obtain a static IP address or set up a physical server at home. The nature of VPS services also ensure that the system administrator will have full control of the virtual machine, meaning that they can configure it however they need to, and manage it freely at any time.

Lastly, the Blade II Online server will make use of Cloudflare's free tier service to provide a vast range of security, quality-of-life, availability and performance improvements for the various network devices that need to be setting up. These include, but are not limited to:

- SSL/TLS connections with valid SSL certificates signed by Cloudflare, allowing any sites proxied by them to appear in users' browsers as "secure", as well as many other advanced SSL features.
- DNS handling, with support for address resolution, mail exchange records and even text files for non-standard verification for things such as DKIM authentication. It also supports DNSSEC[12] and automatic CNAME Flattening[13]. This, along with SSL/TLS is the main reason why Cloudflare will be used to route all traffic - to protect the integrity of online gameplay, those that play Blade II Online, and even the server itself.
- Analytics, with highly detailed stats, graphics and data dumps for examining thing as traffic, security threats, bandwidth usage, and user demographics.

- A firewall with the ability to set up custom block/allow rules.
- Many performance enhancing features such as Auto Minify for JavaScript, CSS and HTML files[14], and automatic response compression[15].
- Edge caching[16], which stores commonly requested files in high performance file servers at the edge of the Cloudflare network, reducing load on the inner webserver and its network.

Elo rating system

While Blade II Online bases its Elo system on the system found in League of Legends, the exact formulae used to calculate Elo in said system have never been publicly released. Therefore, the formulae below are largely based on the standard Elo formula, with some assumptions made based on collective experience playing League of Legends.

There are two main values that must be calculated for each player after a match has been concluded.

- The expected score for the player. This can be described also as the chance that the player had to win the game.
- The new Elo rating for the player.

The formula for calculating the expected score of player A is

$$E_A = \frac{1}{1+10^{(R_A-R_B)/400}},$$

where E_A is the expected score for player A, and R_A and R_B are the current ratings for player A and B respectively.

The constant value of 400 describes the Elo rating difference that would indicate a 10 times greater chance of winning for the target player.

The resultant value is always a value between 0 and 1, making it plain that the formula for calculating the expected score for player B is

$$E_B = 1 - E_A.$$

Therefore, the formula for calculating the new Elo, R'_A, for player A is

$$R'_A = R_A + K(S_A - E_A),$$

where R_A is the current Elo rating for player A, K is the K-factor (akin to the maximum shift in Elo that can occur), and S_A is the actual score for player A.

Thus, the formula for calculating the new Elo, R'_B, for playing B is

$$R'_B = R_B + K(S_B - E_B).$$

The constant K, known as the K-factor, defines the sensitivity to change – how wins and losses impact your Elo rating. Blade II Online will make use of a varying K-factors depending on a players current Elo Rating.

Making use of a smaller K-factor for higher ranked players should in theory avoid Elo inflation at high ratings, where players in the very top percentile will often find themselves playing either players of equal or lower ratings.

Unrated players will have a significantly higher K-factor, with a goal of being able to apply a suitable rating that matches their apparent skill level – The potential for large shifts will quickly give rise to large rating changes should they find themselves winning the majority of their first ten games.

Rating	K-factor
Unrated (less than 10 games played)	100
Less than or equal to 2100 Elo rating	32
Greater than 2100 Elo rating	16

Figure 5 - Blade II Online – K-factor values for Elo rating calculation

The value of S_A is the *actual* score attained by the target player. This is a choice of three constant values (0, 0.5, 1) based on the outcome of the game.

Outcome	Player A score	Player B score
Player A wins, player B loses	1	0
Draw	0.5	0.5
Player A loses, player B wins	0	1

Figure 6 - Blade II Online – Score table for Elo rating calculation

AI opponent

As discussed earlier, Blade II Online will feature an AI opponent versus which players can practice. It will be introduced to all players as an optional training exercise, though it can be skipped if desired.

The aim of this AI is to provide a reliable, relatively predictable opponent that will make logical moves. The aim is not to emulate a human player – in fact, Anguelov *et al.* (2014, p. 5) suggest that 'even some behaviours that actual humans would display should be avoided, because those behaviours *appear* inhuman when performed by an AI-controlled character.'. Rather, the Blade II Online AI opponent aims to provide an experience that players will quickly become accustomed to. This not only creates a smooth learning experience (with no surprises), but also quickly puts players in a position where they will most likely find more stimulation, and hopefully enjoyment by playing games against real opponents. This will hopefully lead to players relying on AI games to practice or strategy testing, as they will be able to effectively disregard the AI opponent as the obstacle between them and a desirable outcome for the game once they reach a point at which they can reliably predict the behaviour of the AI.

The AI will be implemented in two levels, the first of which is an FSM (finite state machine) architecture that shifts between states based on the current state of the match. Anguelov *et al.* (2014, p. 48) explain that 'An FSM breaks down an NPC's overall AI into smaller, discrete pieces known as *states*. Each state represents a specific behaviour or internal configuration, and only one state is considered "active" at a time.'.

By breaking down the logical states of the AI into small, modular state objects, it is possible to create an AI engine that can be repeatedly called from an external source (in this case, most likely as the result of an event listener that is alerted of a change in the game), producing behaviour based on rules written by the implementor that observe the state of a match, and react accordingly.

At its core, the Blade II Online AI's FSM will have three main states – waiting for its turn, determining what move to make, and forwarding its move to the message relay module. Waiting for its turn, and forwarding moves are trivial and will not be covered in this literature review. However, the determination of which move the AI will make is fairly involved and warrants further explanation, along with high level implementation details that are shown below.

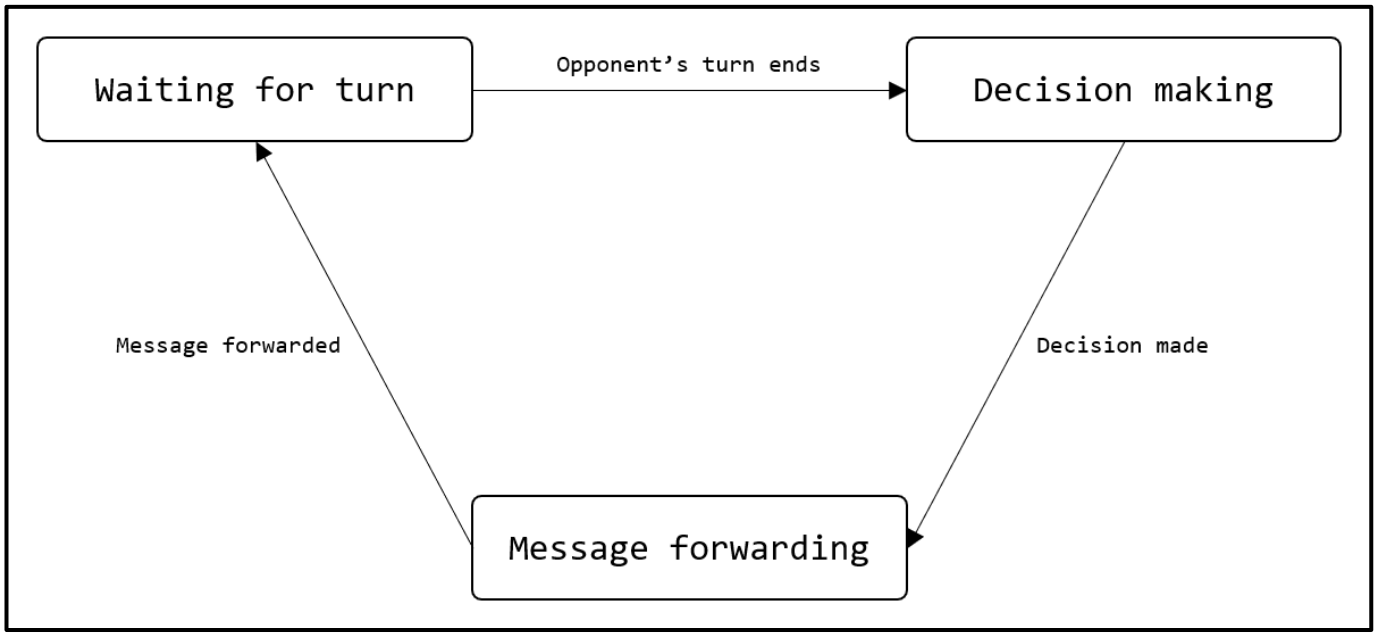

Figure 7 - Blade II Online – AI FSM diagram

The 'decision making' state, responsible for determining what move to make, will implement a depth limited MCTS (Monte Carlo Tree Search) algorithm to determine an appropriate move to make. The algorithm will be depth limited for two reasons – first, to ensure that the AI cannot see too far into the future and therefore cannot reliable determine the best moves, and two, to ensure that the AI does not find itself demanding too much CPU time in order to operate (with the side effect of being quick and responsive, returning play to the human player in real time).

The MCTS algorithm is a modified version of the one described by Cowling *et al.* (2012) and will be implemented with the following logic. Determine all the legal moves based on the current state of the match, by consulting the validation engine.

1. Select a move, with the following bias's:
    a. When there are many cards in the AI's hand, favour lower value cards.
    b. When there are few cards in the AI's hand, favour higher value cards.
    c. Bolt effect cards (select and "flip" one of the cards on the opponent's side of the field) will have a higher priority when there is the option to target a high value card on the opponent's side of the field.
    d. Mirror effect cards (flipping the playing field and scores) will be favoured when the human player's score is considerably higher than the AI's.
    e. Blast effect cards (remove a random card from the opponent's hand) will have a higher priority when the human player has played a majority of lower (< 4) scoring cards and has only a few remaining cards in their hand.
    f. Un-reverse effect cards (un-flip a card flipped by a Bolt card) will have a high priority when the most recent card on the field that was played by the AI is flipped and is of a high (> 5) value.
2. For the selected move, simulate the outcome and calculate a score based on the criteria shown below in figure 8.
3. Using the simulated state of the game from step 3, return to step 1 and test four legal moves, again using the bias's described above. At each tier of the tree, four possible moves are simulated.
4. Repeat steps 1-4, until four moves from the first tier of the tree have been fully explored.

5. For each path, the cumulative scores of each simulation at each tier will yield a value representing the strength of the move. Selecting the initial move with the highest score completes the process.

As can be seen from the methodology above, random selection of moves is avoided at each step to ensure that the AI is both predictable and can reliable make sensible decisions. Moves played by the AI may not be optimal, and this is a conscious design decision to avoid having an AI that is too hard for the average human player to defeat. Having a score of infinite and negative infinite for win/loss states is inspired by Håkonsen's implementation of the calculation for the desirability of a particular state of the game (2015, p. 62).

Outcomes, ordered by desirability	Score
Match win	infinite
Higher score after turn	100 * score difference
Score draw after turn	0
Match draw	-100
Match loss	negative infinite

Figure 8 - Blade II Online – AI MCTS outcome scores

The AI opponent will be served internally by the game client through the same controller as cross-network matches, to avoid adding extra load to the game server, and allowing both the AI and networked matches to use the same input controllers – by routing client updates to the internal AI engine instead, the AI engine will respond as if it received an update from the opposite client.

Development Methodology

Progression

The development of Blade II Online will be guided by the MoSCoW requirements table detailed above in Figure 2, as well as the project Gantt chart (which will be submitted as a separate document). The Gantt chart offers a logical overview of the tasks that will need to be carried out in order to complete the game and the associated dissertation, along with their approximate scheduling and estimated man-days required. During development, the MoSCoW requirements will be implemented in order of priority, according to the current task. For example, if the multiplayer matchmaking queue is completed with time to spare, SSL/TLS encryption will be implemented as well.

Progress will be tracked by following the tasks in the Gantt chart, and by setting up and maintaining a Kanban board to track individual implementation tasks list. A record of work will automatically be curated by the version control software being used for each particular component, during development.

Agile Kanban Methodology

The project will be developed in accordance with the Agile Kanban methodology, and will make use of Trello's free for visualisation of tasks. The agile methodology was largely selected for to its flexibility, and its iterative nature. It's flexibility makes it resilient to any changes that will inevitably be made to the task schedule during development – in the event of a task requiring more/less time than expected, said task, and other tasks can freely be moved around, either clearing up space in a sprint or filling up gaps that have arisen. New tasks can also be slotted into sprints in a similar manner. The iterative nature of Agile encourages developers to revisit existing code, maintaining, refactoring and improving the project as cycles pass. Also, Agile naturally encourages healthy handling of unexpected tasks, such

as bug squashing, allowing for developers to acknowledge and log issues that can be solved in a later sprint if not urgent.

Agile Kanban, an extension of traditional Kanban, is a lean method that places a heavy focus on the visual aspect of organising and maintaining tasks. Team members have the ability to move tasks around by themselves, allowing them to select new tasks from to-do lists and backlogs, whilst also giving them the power to shift completed tasks away for inspection and/or removal.

A study undertaken by Padmanabhan (2018) shows that user responses to Kanban are generally favourable, with many considering it to be 'one of the best agile mechanisms to track the actions (both strategic and run-the-function) in a seamless way.'. Padmanabhan (2018) then goes on to describe how users feel 'empowered due to the full visibility of backlog and the ability to pull the actions to design stage by themselves'. The development of Blade II online will be undertaken by a single developer, with no project manager or authoritative body in place to oversee work progress. The ability to fully utilize a widely recognised and generally well thought of development methodology should empower the developer and reduce the amount of administrative work that must be done themselves.

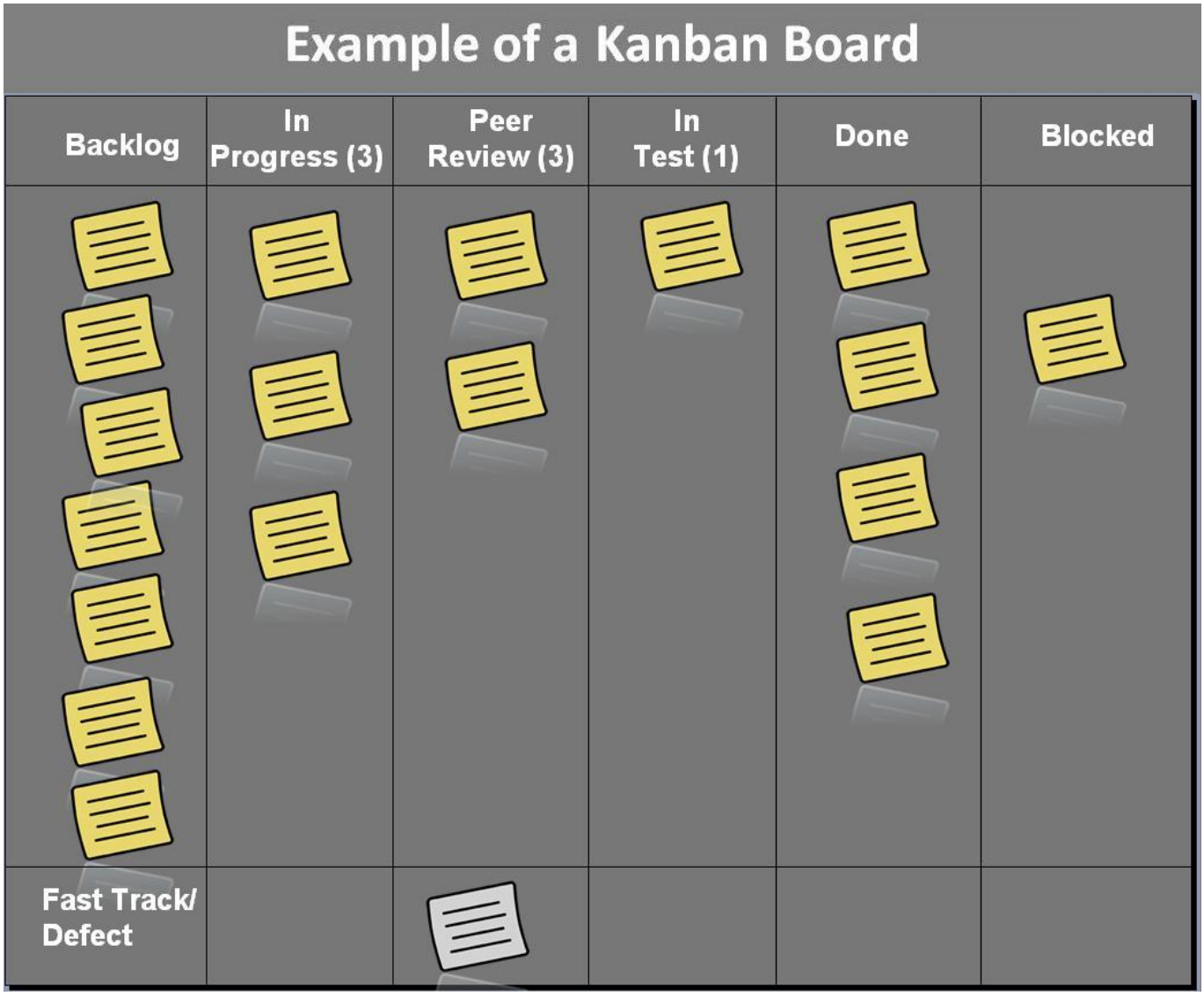

Figure 9 - An example of a Kanban board created by Dr. Ian Mitchell (2014)

Version Control

Development of the game client will be facilitated by a Perforce server running on a Microsoft Azure virtual machine. All other development will be tracked and stored on the Kingston University GitLab server.

Once development is finished, all source code will be ported across to a public GitHub account, so that it can be presented to potential employers via the internet.

Legal, Social and Ethical Considerations

Age Rating and Target Audience

First and foremost, the main target audience for Blade II Online are fans of the Legend of Heroes series. However, the majority of the game's content is focused on the gameplay itself. At its core, Blade II Online is a simple card game, with easy to learn rules and a shallow learning curve. Thematically, and aesthetically, it is being designed for fans of the aforementioned series, but in theory it should be interesting enough for other casual gamers and card game enthusiasts to consider playing.

The age rating for Blade II Online is low for all the ratings agencies that will be contacted for certification. It will feature no content that would match the content descriptors shown described by PEGI (PEGI, no date) such as violence, bad language, or gambling, and should be suitable for players of all ages.

One thing worth noting is that online interactions will not be rated at all. Blade II Online is intended to include a communication channel during matches where players can communicate via preselected messages, similar to Rocket League's quick chat (Khan, 2016). However, it may be taken further with voice communication, which could be used to communicate with the opponent. This system will be optional, however, and in an ideal world would require age verification to enable – though this is out of scope.

Ratings board	Rating
PEGI	3
ESRB	Everyone
CERO	A 全年齢対象 (Zen Nenrei Taishō)

Figure 9 - Blade II Online – Age ratings

Ethical Considerations

Blade II Online will include player avatars which will be selectable from a predetermined set of characters. These characters feature in the Legend of Heroes universe and are included purely for visual purposes. While these will indeed be humanoid characters, there will be no discrimination, abuse, violence, or sexualization portrayed. All other objects in the game are inanimate, and the only interactions that will occur will be the rule-based interactions that occur when cards are played.

Some of the cards feature images of fictional weapons and may be rendered in such a way that they appear to have a physical form in the game. This will be offset by the fact that they are stylized and do not look particularly realistic.

Some of the sound effects will imitate explosions-like sounds or aim to sound like thunder and lightening and other natural phenomenon such as whirlwinds. There is a chance that this could trigger PTSD in certain players, and so the game will offer the option to mute all sound effects.

Intellectual Property

As this game is based on a minigame from select Nihon Falcom games, it will contain IP that belongs to them. In order to ensure that Blade II Online does not infringe on any copyright laws, special attention will be payed to ensure that all usage falls under the 'Non-commercial research and private study' clause described in the Exceptions to copyright guide published by the Intellectual Property Office (2019). Under no circumstances will the game be commercialized, without first either stripping out all the copyrighted content, or after receiving appropriate permission from the owner of the IP.

Due to the non-commercial, educational nature of the game, Unreal Engine 4 will be free to use. 3D models will be created using Maya 2018 under a student license.

Any assets (media, software etc.) from external sources will be used with permission and credited in the third-party licenses section of the in-game settings menu. These will also be detailed thoroughly in the final dissertation. While best efforts will be made to produce these, in the event that it is not possible, third-party assets with an appropriate license will be used, or third-parties will be commissioned to create them. In this event, the third-party will be requested to provide any acknowledgement requests that need to be made, as well as a suitable license to be applied. When used, it will be clearly credited and will be expected to be omitted from assessment.

The music used in the game will also be Nihon Falcom IP. According to 'Falcom's Free Music Use Declaration' it is entirely legal to use any song published by them as long as, amongst other things, the use case is non-commercial and an appropriate accreditation is included in along with the work within which the song is featured (Falcom, 2010).

Sound effects will be created using Bfxr, a free tool with an Apache 2.0 License that can be used to create low-fi sound effects. All sounds created with Bfxr are free to use for any purpose (increpare, not date).

All code written for this project will eventually be uploaded and released under an Apache 2.0 License, and open sourced for the benefit of all.

Data Protection

As Blade II Online will be storing information in persistent storage, it must abide by various data protection laws. When creating an account, users will be informed exactly what data will be stored, and what will be done to said data. In accordance with the Data Protection Act 2018 (Department for Digital, Culture, Media & Sport and Home Office, 2018), special care will be taken to ensure that data is:

1. Used fairly, lawfully and transparently
2. Used for specified, explicit purposes
3. Used in a way that is adequate, relevant and limited to only what is necessary
4. Accurate and, where necessary, kept up to date
5. Kept for no longer than is necessary
6. Handled in a way that ensures appropriate security, including protection against unlawful or unauthorised processing, access, loss, destruction or damage

The sensitive data that will be stored is limited to just an email address, a username, and a password. The email address will be required for account validation and password resets/recovery, whilst the username and password will be used for authentication, and as a display name in matches and in leaderboards.

References

Anguelov, B. *et al.* (2014) *Game AI Pro*. Edited by S. Rabin *et al.* Boca Raton, FL: CRC Press.

Bizreach (2018) *Puroguramingu Gengobetsu Nenshuu Chuuōchi Wo Happyō, Kyuujin Kensaku Enjin「Sutanbai」Shirabe*. Available at: https://www.bizreach.co.jp/pressroom/pressrelease/2018/0807.html (Accessed: 22 October 2019).

Blevins, T. (2000) *Magic: The Gathering Review*. Available at: https://www.gamespot.com/reviews/magic-the-gathering-review/1900-2542422/ (Accessed: 27 September 2019).

Blizzard Entertainment (2019) *Lightning Storm*. 16 May. Available at: https://d2q63o9r0h0ohi.cloudfront.net/videos/npe/videos/LightningStorm-final-be597a7d03990bcf124753dcb4f60ba29c232ee531687c34d960aff589e40b8361e2026e742ac3562cc956f3fdf11c6673da087660beaee9ddc1ad9cc1f53147.mp4 (Accessed: 2 October 2019).

Bregger, P. (2019) *Chron X (1997) Windows release date*. Available at: https://www.mobygames.com/game/windows/chron-x/release-info/ (Accessed: 27 September 2019).

Brode, B. [@bbrode] (2013) @Ziethriel mixed - projected textures on 3d models [Twitter] 14 December. Available at: https://twitter.com/bbrode/status/411927244827684866/ (Accessed: 2 October 2019).

Brode, B., Chayes J. (2013) Interviewed by Miaari for *Blizz Planet*, 24 March. Available at: https://www.youtube.com/watch?v=FRVSsVCC5v8 (Accessed: 2 October 2019).

Caldwell, J. (2019) *Microsoft Solitaire enters the World Video Game Hall of Fame*. Available at: https://www.onmsft.com/news/microsoft-solitaire-enters-the-world-video-game-hall-of-fame/ (Accessed: 27 September 2019).

Cowling, P. et al. (2012) 'Ensemble Determinization in Monte Carlo Tree Search for the Imperfect Information Card Game Magic: The Gathering', *IEEE Transactions on Computational Intelligence and AI in Games*, 4(4), p. 243. doi:10.1109/TCIAIG.2012.2204883.

Department for Digital, Culture, Media & Sport and Home Office (2018) *Data Protection Act 2018*. Available at: https://www.gov.uk/government/collections/data-protection-act-2018 (Accessed: 25 October 2019).

Duncan, I. (2019) *Build a RESTful JSON API with Golang*. Available at: https://medium.com/the-andela-way/build-a-restful-json-api-with-golang-85a83420c9da (Accessed 05 October 2019).

Ellingwood, J. (2017) *How to create a Self-Signed SSL Certificate for Nginx on CentOS 7*. Available at: https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-nginx-on-centos-7/ (Accessed: 23 October 2019).

Famitsu (2010) *Epic Games No Nihon Jōriku Ga Kettei, Unreal Engine No Nihongo Sapōto Wo Kyōka*. Available at: https://automaton-media.com/articles/interviewsjp/20180524-68476/ (Accessed: 22 October 2019).

Falcom (2010) *Falcomu Ongaku Furii Sengen - Gakkyoku Riyō Kiyaku*. Available at: https://www.falcom.co.jp/music-use/rules (Accessed: 25 October 2019).

Glazer, J. and Madhav, S. (2016) *Multiplayer Game Programming*. Crawfordsville, IN: Pearson. The Addison-Wesley Game Design and Development Series.

go-chi/chi: lightweight, idiomatic and composable router for building Go HTTP services (2019) Available at: https://github.com/go-chi/chi/tree/master/ (Accessed: 22 October 2019).

Google (no date) *Package json*. Available at: https://golang.org/pkg/encoding/json/#Unmarshal (Accessed 23 October 2019).

Håkonsen, E. (2015) *System-and AI design of a digital collectible card game*. PhD thesis. University of Copenhagen. Available at: https://www.pdf-archive.com/2017/12/20/hearthstone-ai-thesis-ejnar-h-konsen/hearthstone-ai-thesis-ejnar-h-konsen.pdf (Accessed: 24 October 2019).

Hearthstone (2017) *Behind the Card | Amara: Warden of Hope*. 13 April. Available at: https://www.youtube.com/watch?v=l0qqFROfHWE (Accessed: 2 October 2019).

Hiwarale, U. (2018) *Achieving concurrency in Go*. Available at: https://medium.com/rungo/achieving-concurrency-in-go-3f84cbf870ca/ (Accessed: 22 October 2019).

IGN (2014) *Card Rarity - Hearthstone: Heroes of WarCraft Wiki Guide*. Available at: https://uk.ign.com/wikis/hearthstone-heroes-of-warcraft/Card_Rarity (Accessed: 23 October 2019).

increpare (no date) *Bfxr. Make sound effects for your games.* Available at: https://www.bfxr.net/ (Accessed: 25 October 2019).

Intellectual Property Office (2019) *Exceptions to copyright*. Available at: https://www.gov.uk/guidance/exceptions-to-copyright#non-commercial-research-and-private-study (Accessed 25 October 2019).

Jebens, H. (1998) *Sanctum Open for Business*. Available at: https://www.gamespot.com/articles/sanctum-open-for-business/1100-2464321/ (Accessed: 27 September 2019).

Johnsson, H. (2019) *REST-API with Golang, Mux and MySQL*. Available at: https://medium.com/@hugo.bjarred/rest-api-with-golang-mux-mysql-c5915347fa5b (Accessed: 23 October 2019).

Kawasaki, T. (2013) 'Interview with Taka KAWASAKI'. Interview with Taka Kawasaki. Interviewed by J. Szczepaniak for *The Untold History of Japanese Game Developers Volume 2*, 31 October, p. 391.

Kawasaki, T. (2017) Interviewed by Keigo Toyoda for *Famitsu*, 6 September. Available at: https://www.famitsu.com/news/201709/06140985.html (Accessed: 23 October 2019).

Khan, I. (2016) *OMG! Wow! Rocket League quick chat options expanding*. Available at: https://www.digitaltrends.com/gaming/rocket-league-quick-chat-update/ (Accessed 25 October 2019).

Kollar, P. (2015) *Play Hearthstone on your iPhone or Android phone starting today*. Available at: https://www.polygon.com/2015/4/14/8407107/hearthstone-ios-android-iphone-release/ (Accessed: 27 September 2019).

Krogh-Jacobsen, T. (2018) *2018.2 is now available*. Available at: https://blogs.unity3d.com/2018/07/10/2018-2-is-now-available/ (Accessed 23 October 2019).

Lima Beans (2009) *Re: Curious: What Programming Language?*. [Online discussion group] 7 October. Available at: http://forums.na.leagueoflegends.com/board/showthread.php?t=16318 (Accessed: 4 October 2019).

Merril, M. (2009) Interviewed by Suzie Ford for *WarCry Network*, 24 January. Available at: https://web.archive.org/web/20131213123923/http://www.warcry.com/articles/view/interviews/5686-League-of-Legends-Marc-Merrill-Q-A/ (Accessed: 2 Oct. 2019).

Mitchell, I. (2014) *Example of a Kanban Board separating internally and externally blocked work*. Available at: https://upload.wikimedia.org/wikipedia/commons/thumb/7/7b/Kanbanboardidentifyinginternalexternalblockages.jpg/536px-Kanbanboardidentifyinginternalexternalblockages.jpg (Accessed: 25 October 2019).

Nintendo (1998) *Pokémon Kaado GB*. Available at: https://www.nintendo.co.jp/n02/dmg/acxj/index.html (Accessed: 27 September 2019).

Paah (2010) *HoN/LoL release dates?*. [Online discussion group] 12 July. Available at: http://forums.euw.leagueoflegends.com/board/showthread.php?t=78609 (Accessed: 2 October 2019).

Padmanabhan, V. (2018) 'Functional Strategy Implementation - Experimental Study on Agile KANBAN', *Sumedha Journal of Management*, 7(2), pp. 6-17. Available at: https://search-proquest-com.ezproxy.kingston.ac.uk/docview/2149601807?accountid=14557 (Accessed 25 October 2019).

PEGI (no date) *What do the labels mean?*. Available at: https://pegi.info/what-do-the-labels-mean (Accessed: 24 October 2019).

Rawdat, A. (2017) *Rate Limiting with NGINX and NGINX Plus*. Available at: https://www.nginx.com/blog/rate-limiting-nginx/ (Accessed: 23 October 2019).

Suzuki, G. (2014) *Unreal Engine Pōtaru Saito Ga Kōshiki Nihongoka, Unreal Editor Honyaku Mo Shinkōchuu.* Available at: https://www.gamespark.jp/article/2014/05/01/48231.html (Accessed: 23 October 2019).

Sweeney, T. (2018) Interviewed by Minoru Umise for *Automaton Media*, 24 May. Available at: https://automaton-media.com/articles/interviewsjp/20180524-68476/ (Accessed: 22 October 2019).

Sookocheff, K. (2019) *How Do Websockets Work?*. Available at: https://sookocheff.com/post/networking/how-do-websockets-work/ (Accessed: 23 October 2019).

Warren, M. (2019) *Is C# a low-level language?*. Available at: https://mattwarren.org/2019/03/01/Is-CSharp-a-low-level-language/ (Accessed 22 October 2019).

Wilson, J. (2014) *Even Hearthstone runs on Unity — and that's why it's already on iPad*. Available at: https://venturebeat.com/2014/04/24/even-hearthstone-runs-on-unity-and-thats-why-its-already-on-ipad/ (Accessed: 27 September 2019).

Wu, L. (2018) *Setting-up mitmproxy on macOS to intercept https requests*. Available at: https://medium.com/what-i-talk-about-when-i-talk-about-ios-developmen/setting-up-mitmproxy-on-macos-to-intercept-https-requests-f3cba29ff003 (Accessed: 23 October 2019).

Yin-Poole, W. (2014) *Hearthstone leaves beta and launches proper on PC and Mac*. Available at: https://www.eurogamer.net/articles/2014-03-12-hearthstone-leaves-beta-and-launches-proper-on-pc-and-mac (Accessed: 3 September 2019).

Zeriyah (2014) *Welcome to the Hearthstone Launch!*. Available at: https://playhearthstone.com/en-us/blog/13154923/ (Accessed: 3 October 2019).

Zwicker, J. [@Zwickarr] (2013) @Ziethriel @bdbrode I take a painting made by Ben Thompson and project it on 3d models. Models are a bit distorted :) [Twitter] 14 December. Available at: https://twitter.com/Zwickarr/status/411939154990071808/ (Accessed: 3 October 2019).

Appendix

Gantt chart, as a PDF, submitted with the name **k1611060-project-proposal-gantt-chart.pdf**.

Footnotes

[1] Nihon Falcom is a Japanese Game Developer, established in 1981, famous for its RPG's. More at: https://www.falcom.co.jp/ (Available only in Japanese)..

[2] Trails of Cold Steel II is a story RPG developed by Nihon Falcom. More at: https://www.falcom.co.jp/sen2/kai/ (Available only in Japanese)..

[3] Tokyo Xanadu eX+ is an ARPG developed by Nihon Falcom. More at: https://www.falcom.co.jp/txana_explus/ (Available only in Japanese)..

[4] This is a gameplay screenshot taken on the 2nd of October 2019. It captures the action from a game of ARAM on the League of Legends EUW server.

[5] Bizreach is a Japanese recruitment agency. More at: https://www.bizreach.jp/ (Available only in Japanese).

[6] GORM is an open-source ORM library for Go, providing a layer of abstraction when interacting with a database. More at: https://gorm.io/.

[7] The Blade II Online game server will be hosted on a VPS with 1 virtual CPU of unknown speed, and 1GB of shared RAM. The network speed is limited to 100MB/s in both directions. The game server implementation will take into consideration the limited resources available.

[8] mitmproxy is a free, open source HTTPS proxy that can be used to manipulate HTTP traffic in real-time. More at: https://mitmproxy.org/.

[9] Nginx is a free, open-source, lightweight web server, which also offers reverse proxying, load balancing, HTTP caching and more. More at: https://www.nginx.com/resources/wiki/.

[10] A virtual private server (VPS) is a virtual machine hosted by a company, which offers access to it as a service. They are virtualized machines running in an isolated environment, as part of a larger virtual or physical machine. More at: https://en.wikipedia.org/wiki/Virtual_private_server/.

[11] Cloudflare is one of the largest networks on the internet, and offers services such as DNS, DDOS protection, SSL certificates, and various performance enhancers such as edge caching. These services are offered for free to small scale users. More at: https://www.cloudflare.com/learning/what-is-cloudflare/.

[12] DNSSEC offers protection against forged DNS query responses. More at: https://support.cloudflare.com/hc/en-us/articles/360006660072/.

[13] CNAME flattening allows CNAME records to be created for a domains root address, whilst still respecting DNS specifications. More at: https://support.cloudflare.com/hc/en-us/articles/200169056-Understand-and-configure-CNAME-Flattening/.

[14] Auto Minify is the product name for Cloudflare's Minification service, which minimizes the size of code and markup files that are served by them. More at: https://www.imperva.com/learn/performance/minification/.

[15] Cloudflare offers Brotli, which speeds up page load times for data returned via HTTPS by compressing responses. More at: https://support.cloudflare.com/hc/en-us/articles/200168396/.

[16] Edge caching is the process by which frequently requested content such as media or scripts is stored on, and served directly from, servers at the edge of the network that serves the content. The "Edge" can be described as the outermost layer of connectivity – which may for example form the bridge between the private network and the outside internet. More at: https://www.cloudflare.com/learning/cdn/glossary/edge-server/.