

Universidad ORT Uruguay  
Facultad de Ingeniería

# Hoodies: Turn Based Strategy Card Game

Entregado como requisito para la obtención del título de Analista Programador

Agustín Acosta - 212428  
Ramiro Cardinal- 216596  
Manuel De Armas - 174321

Tutor: Santiago Fagnoni

2022

## Declaración de Autoría

Nosotros; Agustín Acosta, Ramiro Cardinal y Manuel De Armas. Declaramos que el trabajo que se presenta en esta obra es de nuestra propia mano.

Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el proyecto final de la carrera Analista Programador.
- Cuando hemos consultado trabajos publicados por otros, lo hemos atribuido con claridad.
- Cuando hemos citado obras de otros autores, hemos indicado las fuentes. Con excepción de dichas citas, la otra es enteramente nuestra.
- En la obra, hemos acusado recibo de las ayudas recibidas.
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros.
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

		
Agustín Acosta	Ramiro Cardinal	Manuel De Armas

## Agradecimientos

A Santiago Fagnoni, nuestro tutor, por su experiencia y orientación, que a lo largo del proyecto nos mantuvo enfocados en las cosas que eran importantes para arribar a un producto final de calidad que refleje nuestras intenciones, conocimientos y que esté alineado con lo que requiere la universidad.

Y a nuestras familias, amigos, colegas y compañeros que siempre estuvieron dispuestos a apoyar desde sus capacidades en maneras a veces invisibles pero cruciales para permitirnos llevar a cabo el proyecto.

## Abstract

Este documento detalla el proceso de desarrollo de un videojuego competitivo en linea dentro del genero de juegos de estrategia por turnos basados en cartas con posibilidades de monetización que lo hagan atractivo a inversores y que no alienen a potenciales jugadores.

El objetivo principal es crear un corte vertical o prueba de concepto que demuestre la posibilidad de convertir nuestro resultado final, en un producto de entretenimiento de calidad que cumpla las características antes descriptas.

Para cumplir con los objetivos trazados nos propusimos desarrollar una aplicación de escritorio para PC usando el motor de videojuegos Unity para la interacción de los jugadores y Amazon Web Services para la infraestructura de servicios online y persistencias de datos.

# Palabras Clave

Unity

Game Engine

Amazon Web Services

AWS GameLift

Videojuego

C#

Duelyst

Linux

AWS Lambda

AWS DynamoDB

Jira

Confluence

Plastic SCM

Hoodies

AWS Cognito

Juego

Estrategia

# Índice

<b>1 Anteproyecto.....</b>	<b>11</b>
1.1 Introducción.....	11
1.1.1 Contexto .....	11
1.1.2 Motivación .....	12
1.1.3 Code Name .....	12
1.2 Integrantes y roles .....	12
1.2.1 Productor del juego .....	12
1.2.2 Diseñador del juego .....	12
1.2.3 Control de Calidad .....	13
1.2.4 Desarrollador .....	13
1.3 FODA .....	13
1.3.1 Fortalezas .....	13
1.3.2 Oportunidades .....	13
1.3.3 Debilidades.....	14
1.3.4 Amenazas .....	14
1.4 Riesgos.....	14
1.4.1 Análisis de Riesgos .....	14
1.5 Plan de Proyecto .....	17
1.5.1 Presentación del Grupo Foco .....	17
1.5.2 Herramientas y Tecnologías.....	18
1.5.3 Estrategia de Control de Calidad .....	21
1.5.4 Método de trabajo.....	26
1.5.5 Cronograma .....	27
1.6 Requerimientos Funcionales.....	29
1.7 Requerimientos No Funcionales .....	34
1.8 Pre-Producción .....	35
1.8.1 Stack de Tecnologías .....	35
1.8.2 Diagramas de Clases .....	36
1.8.3 Flujo de Pantallas.....	38
1.8.4 Mockups de Pantallas .....	40
1.8.5 Arte Conceptual de Gameplay.....	45

# Índice

<b>2 Proyecto .....</b>	<b>47</b>
<b>2.1 Sprint 1 .....</b>	<b>47</b>
<b>2.1.1 Resumen.....</b>	<b>47</b>
<b>2.1.2 Tareas .....</b>	<b>47</b>
<b>2.1.3 Investigación .....</b>	<b>48</b>
<b>2.1.4 Investigación - Servicio de alojamiento de servidor .....</b>	<b>48</b>
<b>2.1.5 Investigación - Servicio de manejo de usuarios .....</b>	<b>48</b>
<b>2.1.6 Investigación - Herramienta de control de versión.....</b>	<b>48</b>
<b>2.1.7 Diseño - Descripción del Gameplay.....</b>	<b>49</b>
<b>2.1.8 Implementación - Control de Versión .....</b>	<b>52</b>
<b>2.2 Sprint 2 .....</b>	<b>52</b>
<b>2.2.1 Resumen.....</b>	<b>52</b>
<b>2.2.2 Tareas .....</b>	<b>53</b>
<b>2.2.3 Investigación .....</b>	<b>53</b>
<b>2.2.4 Investigación - Comunicación entre servicios .....</b>	<b>53</b>
<b>2.2.5 Investigación - Persistencia de Datos .....</b>	<b>53</b>
<b>2.2.6 Implementación - Game Server.....</b>	<b>56</b>
<b>2.2.7 Implementación - Login y Register .....</b>	<b>57</b>
<b>2.2.8 Stack de Tecnologías .....</b>	<b>59</b>
<b>2.2.9 Gestión de Proyecto.....</b>	<b>60</b>
<b>2.3 Sprint 3 .....</b>	<b>60</b>
<b>2.3.1 Resumen.....</b>	<b>60</b>
<b>2.3.2 Tareas .....</b>	<b>61</b>
<b>2.3.3 Implementación - Producción de Documentación Interna .....</b>	<b>61</b>
<b>2.3.4 Implementación - Lobby Público y Privado.....</b>	<b>62</b>
<b>2.3.5 Implementación - Arquitectura de Sonido .....</b>	<b>65</b>
<b>2.3.6 Implementación - Arquitectura de escenas.....</b>	<b>68</b>
<b>2.4 Sprint 4 .....</b>	<b>69</b>
<b>2.4.1 Resumen.....</b>	<b>69</b>
<b>2.4.2 Tareas .....</b>	<b>69</b>
<b>2.4.3 Implementación - Estructura de UI.....</b>	<b>70</b>
<b>2.4.4 Implementación - Login Scene.....</b>	<b>72</b>
<b>2.4.5 Implementación - Menu Scene.....</b>	<b>74</b>

# Índice

2.4.6 Investigación - Integración AWS Lambda y .NET .....	79
<b>2.5 Sprint 5 .....</b>	<b>84</b>
2.5.1 Resumen .....	84
2.5.2 Tareas .....	85
2.5.3 Implementación - Match Scene.....	86
2.5.4 Implementación - Código Compartido Entre Cliente y Servidor .....	86
2.5.5 Implementación - Clases de Gameplay .....	88
2.5.6 Implementación - GameLift local.....	88
2.5.7 Investigación - AWS Roles and Policies.....	88
2.5.8 Investigación - API Gateway con Lambda.....	92
<b>2.6 Sprint 6 .....</b>	<b>96</b>
2.6.1 Resumen .....	96
2.6.2 Tareas .....	96
2.6.3 Re-Evaluación de GameLift Local.....	97
2.6.4 Consolidación de AWS Account.....	98
2.6.5 Lógica Auxiliar del Tablero .....	99
2.6.6 Lógica de Animaciones .....	102
<b>2.7 Finalización 1.....</b>	<b>105</b>
2.7.1 Re-Estructuración de Ambiente de Desarrollo .....	105
2.7.2 Otras Tareas de Refactorización .....	107
2.7.3 Desarrollo del Gameplay .....	113
<b>2.8 Finalización 2.....</b>	<b>133</b>
2.8.1 Esfuerzo de Integración .....	133
2.8.2 Comunicación Entre Clientes y Servidor .....	134
<b>2.9 Finalización 3.....</b>	<b>145</b>
2.9.1 Administración de Tareas con Contingencias .....	145
2.9.2 Command Line Utility .....	148
2.9.3 Local Game Server .....	150
2.9.4 Manejo de Excepciones en Servidor.....	152
2.9.5 Evolución del Aspecto Visual del Juego.....	153
2.9.6 Sincronización Servidor-Cliente de los Datos Iniciales de una Partida .....	156
<b>3 Evolución de Stack de Tecnologías.....</b>	<b>160</b>
3.1 Versión 1 .....	160

# Índice

3.2 Versión 2 .....	161
<b>4 Conclusiones .....</b>	<b>162</b>
4.1 Introducción.....	162
4.1.1 Meta Inicial .....	162
4.1.2 Producto Final.....	162
4.2 Requerimientos.....	163
4.2.1 Requerimientos Funcionales.....	163
4.2.2 Requerimientos No Funcionales .....	166
4.3 Riesgos.....	166
4.3.1 Manifestación de Riesgos .....	167
4.3.2 Grupo Foco .....	167
4.4 Herramientas y Tecnologías.....	168
4.4.1 Valoración de Herramientas Utilizadas .....	168
4.4.2 Stack de Tecnologías .....	169
4.5 Control de Calidad .....	170
4.6 Método de Trabajo .....	171
4.7 Posibles Mejoras .....	171
4.7.1 Mejoras a la Presentación.....	171
4.7.2 My Decks .....	171
4.7.3 Métricas .....	171
4.7.4 Profundizar la Jugabilidad .....	172
4.8 Lecciones Aprendidas.....	172
4.8.1 Estimaciones .....	172
4.8.2 Herramientas.....	172
4.8.3 Gestión.....	172
4.8.4 Colaboración .....	172
4.9 Cierre .....	173
<b>5 Glosario.....</b>	<b>174</b>
<b>6 Bibliografía .....</b>	<b>175</b>
6.1 Vínculos Externos.....	175
6.2 Referencias Bibliográficas .....	175

# Índice

7 Anexos .....	177
7.1 Manual de Usuario .....	177
7.1.1 Hoodies.....	177

# 1 Anteproyecto

## 1.1 Introducción

### 1.1.1 Contexto

Al momento de escribir este documento, el mercado altamente popular de juegos dentro del género *Digital collectible card game*, DCCG [g1] por su sigla en inglés, está dividido entre pocos exponentes. Hoy en día, el mayor exponente es el juego denominado *Hearthstone* [v1] con aproximadamente 300 mil jugadores por día. Este género, que busca emular los juegos tradicionales de cartas colecciónables, por lo general está enfocado en la jugabilidad en línea y otras veces en la jugabilidad local contra el ordenador. Surgió a fines de los 90 con exponentes tales como *Magic: The Gathering* [v2], *Pokémon Trading Card Game* [v3] y *Yu-Gi-Oh! Duel Monsters* [v4], con la intención de replicar juegos de cartas colecciónables físicos. Es por esto que, por lo general, mantienen características similares como la existencia de distintos jugadores que tienen puntos de vida, un mazo de cartas y una mano de cartas que se llena en base a este último. Las cartas pueden representar criaturas con capacidades y propiedades como por ejemplo: ataque, puntos de vida, defensa, efectos. Otras veces representan hechizos que pueden afectar tanto al rival como a uno mismo, por ejemplo, permitiendo o no tomar más cartas del mazo, dañar al enemigo o curar a un aliado, las posibilidades parecen ser infinitas.

A esto se le suma un campo de batalla basado en cuadrantes o celdas y la interacción del jugador con éste, algo más asociado al género de *Turn Based Strategy*, TBS [g2]. Este género milenario se compone de juegos como el Ajedrez [v5] o las Damas [v6], pero en términos de video juegos, existen series con gran popularidad como *Civilization* [v7] y *XCOM 2* [v8].

El atractivo de estos juegos nace, en parte, de la posibilidad de combinar estas cartas y acciones utilizando el ingenio para derrotar al contrincante. Pero no es solo al momento de jugar donde se debe aplicar estrategia, sino que también al momento de construir el mazo, previo al comienzo de la partida se deben seleccionar las cartas que se desea que formen parte de la misma, es importante que las cartas hagan sinergia entre si y puedan ser combinadas o sean elegidas para contrarrestar a un oponente específico.

Esto, además de un atractivo desde el punto de vista lúdico, es una gran avenida de monetización para los desarrolladores e inversores de los juegos. Las cartas, en su gran mayoría, se obtienen mediante paquetes o sobres que tienen un costo, ya sea en moneda virtual del propio juego o en dinero real e incluso se forman grandes mercados secundarios que mantienen viva a la comunidad que participa en el juego.

### 1.1.2 Motivación

La motivación de este proyecto consiste en la creación de un videojuego que cumpla con las características de un juego *DCCG* que permita capturar una porción de la audiencia.

La propuesta del equipo consiste en construir un juego 1 vs 1 de estrategia por turnos en línea con una presentación 2.5D [g3]. La partida se desarrollará en un tablero con celdas, dónde cada jugador contará con un mazo creado previamente, una mano de cartas que se nutre del mismo y un capitán con una casilla asignada al inicio, cuyos puntos de vida son los mismos que posee el jugador, es decir que si estos llegan a 0 el jugador perderá la partida. El objetivo es derrotar al capitán enemigo. Para conseguirlo, mediante cartas que tendrán un costo de uso, se podrá invocar unidades en el tablero y lanzar hechizos (“tecnologías” dentro del juego). Las unidades tendrán acciones tales como movimiento y ataque, permitiendo dañar a las unidades del enemigo y a su capitán. Los hechizos ayudarán a controlar el tablero, robar cartas del mazo y dañar al enemigo.

### 1.1.3 Code Name

En el desarrollo de este documento se verá en repetidas ocasiones el nombre Hoodies, HOOD o variaciones de lo mismo. Vale aclarar que esto es un nombre código, elegido al principio del proyecto antes de tener claro como se iba a desarrollar, tanto temáticamente como visualmente.

## 1.2 Integrantes y roles

### 1.2.1 Productor del juego

Será el encargado de mantener la visión del videojuego que se estableció desde el principio. Velará por el cumplimiento de las distintas etapas del proyecto y los plazos establecidos. Manuel De Armas, Agustín Acosta y Ramiro Cardinal tendrán este rol.

### 1.2.2 Diseñador del juego

Este rol comprende todo lo que a diseño se refiere, diseñar la temática del juego, el arte que este lleva, diseño de niveles, sistema de reglas, equilibrio del juego y mecánicas. Este rol lo cumplirán Manuel De Armas, Agustín Acosta y Ramiro Cardinal.

### 1.2.3 Control de Calidad

La función de este rol no consiste solamente en encontrar errores, sino que también consiste en proveer comentarios, críticas constructivas sobre cualquier aspecto del juego. Este rol en primer instancia estará cubierto por Manuel De Armas, Agustín Acosta y Ramiro Cardinal, pero quienes toman con más importancia este rol será el grupo foco, que está formado por profesionales en el rubro de videojuegos, y también usuarios no profesionales voluntarios, quienes aportarán una visión mas allegada un usuario final mediante encuestas.

### 1.2.4 Desarrollador

La función de este rol es claramente desarrollar el código fuente del juego para que cumpla con todos los requerimientos definidos en etapas previas. Este rol lo cumplirán Manuel De Armas, Agustín Acosta y Ramiro Cardinal.

## 1.3 FODA

Dada la naturaleza del proyecto, del equipo y el contexto en el que se trabaja, es posible realizar el siguiente análisis.

### 1.3.1 Fortalezas

- Se encuentran disponibles grandes cantidades de tutoriales y guías online de alta calidad, tanto pagas como gratis, que podrán ayudar a un mejor y más eficiente proceso de desarrollo.
- El tutor de la Universidad ORT encargado de nuestro proyecto tiene experiencia en el área del desarrollo de videojuegos, por lo que su tutoría será de gran valor para el grupo.
- Asimismo, todos los miembros del grupo tienen interés en esta área, algo que favorece al proceso de aprendizaje.

### 1.3.2 Oportunidades

- Actualmente no existen competidores de envergadura en el género DCCG/TBS.

### 1.3.3 Debilidades

- Al tratarse de un videojuego, será necesario usar herramientas y técnicas de programación no vistas en la carrera. Muchas partes del proyecto deberán ser aprendidas durante su desarrollo.
- Falta de experiencia liderando y planificando proyectos.

### 1.3.4 Amenazas

- El surgimiento de competidores durante el desarrollo del proyecto.
- Que se haya sobre-estimado la cantidad de usuarios potencialmente interesados en el juego.
- Que surja la necesidad de cambiar los requerimientos del proyecto abruptamente durante su desarrollo, por falta de experiencia, o por factores difíciles de predecir.

## 1.4 Riesgos

### 1.4.1 Análisis de Riesgos

Los riesgos identificados no son muchos, pero pueden ser de gran impacto si no existen estrategias de mitigación y/o contingencia para ellos.

ID Riesgo	R01
Descripción	<p><u>Necesidad de uso de nuevas tecnologías y herramientas</u></p> <p>Dada la naturaleza del proyecto a realizar, los estudiantes deberán usar nuevas herramientas y tecnologías — explicadas en otras partes del documento. Esto no sólo conlleva un proceso de familiarización, sino que también el aprendizaje de técnicas y patrones dentro de las herramientas mismas.</p> <p>Otro factor a considerar es que todos los estudiantes poseen distintos grados de familiarización con las herramientas, por lo que un proceso de nivelación podría ser necesario.</p>

<b>Categoría</b>	Técnico
<b>Probabilidad</b>	Alto
<b>Impacto</b>	Alto
<b>Frecuencia de control</b>	Semanal
<b>Plan de mitigación</b>	Contacto constante entre los integrantes del grupo, y con el tutor asignado para compartir información. Uso de tutoriales online para profundizar conocimientos.
<b>Plan de contingencia</b>	En caso de existir momentos donde el progreso del proyecto se vea comprometido, los integrantes podrán ayudarse entre sí — con ayuda del tutor en caso de ser necesario.

Tabla 1. Riesgo R01

<b>ID Riesgo</b>	R02
<b>Descripción</b>	<p><u>Mala estimación de los tiempos de desarrollo</u></p> <p>Como fue mencionado en el punto anterior, los integrantes del grupo no poseen un gran nivel de familiaridad con muchas de las herramientas a utilizar. Esto implica, entonces, que el grupo puede tener cierta dificultad en estimar cuánto tardarán algunas tareas. Esto puede ser para el beneficio o el detrimento del proyecto dependiendo si se sobreestima o subestima el tiempo.</p>
<b>Categoría</b>	Gestión
<b>Probabilidad</b>	Medio
<b>Impacto</b>	Alto
<b>Frecuencia de control</b>	Semanal

<b>Plan de mitigación</b>	Monitoreo del progreso de los sprints durante los mismos por parte de los integrantes.
<b>Plan de contingencia</b>	Re-estimación de los tiempos y sprints pautados. Priorizando algunos de ser necesario. En casos críticos, se podrán modificar los objetivos del proyecto.

Tabla 2. Riesgo R02

<b>ID Riesgo</b>	R03
<b>Descripción</b>	<u>Error en elección de la herramienta a usar</u> La herramienta elegida para el funcionamiento en línea del juego puede no ser la adecuada para las necesidades del proyecto.
<b>Categoría</b>	Técnico
<b>Probabilidad</b>	Baja
<b>Impacto</b>	Alto
<b>Frecuencia de control</b>	Instancia única de prueba
<b>Plan de mitigación</b>	El equipo ya hizo un proceso de investigación para determinar la herramienta a usar.
<b>Plan de contingencia</b>	El equipo determinó qué alternativa usar en caso de ser necesario y migrar todo a la misma.

Tabla 3. Riesgo R03

<b>ID Riesgo</b>	R04
------------------	-----

<b>Descripción</b>	<u>Indisponibilidad de alguno/s de los integrantes</u>  La planificación del proyecto asume que todos los desarrolladores estarán disponibles en todo el transcurso del mismo. Sin embargo, existe la posibilidad que algunos de los integrantes estén indiscretos o indisponibles por motivo de viaje o causa mayor.
<b>Categoría</b>	Personal
<b>Probabilidad</b>	Media
<b>Impacto</b>	Medio
<b>Plan de mitigación</b>	A excepción de casos de viaje, donde un integrante puede dar aviso previo al resto, la naturaleza fortuita de este riesgo implica que no será posible de mitigar.
<b>Plan de contingencia</b>	Como el proyecto no se puede posponer, el equipo tendrá la elección de dedicarle más horas al proyecto para cumplir las metas, o en casos más críticos, de reducir el alcance.

Tabla 4. Riesgo R04

## 1.5 Plan de Proyecto

### 1.5.1 Presentación del Grupo Foco

En vista de que la motivación de nuestro proyecto surge del equipo de proyecto mismo, vemos como necesario la definición de un grupo de individuos u organizaciones, independientes al equipo que puedan actuar como validadores de calidad y efectividad del producto generado.

Los integrantes de este grupo se dividen en subgrupos ya que para obtener información de utilidad es importante presentar versiones del producto que se ajusten a la capacidad evaluativa que puedan hacer los integrantes cada subgrupo.

### 1.5.1.1 Subgrupo 1 - Profesionales

Dentro de este subgrupo se encuentran trabajadores del ámbito del desarrollo de software y videojuegos. Entendemos que este subgrupo puede apreciar con una visión critica el producto en estados de desarrollo temprano, con muchas piezas faltantes o en estado de incompletitud. Recibirán versiones ejecutables con un menor nivel de pulido que un usuario final. Asistirán a reuniones informales por teleconferencia o comunicación por email para entregar sus devoluciones.

#### Integrantes

- *ARF Games Studio* [v9] - Estudio de desarrollo de videojuegos local con gran trayectoria trabajando para clientes del exterior construyendo juegos web y aplicaciones móviles.
- *Real Virtuality Games* [v10] - Estudio independiente local de desarrollo de videojuegos 3D, tienen una marcada experiencia trabajando en *Unity Game Engine* [v11]
- Manuel Atienza [v12]
- - Desarrollador de software local con experiencia dentro del genero de los DCCG

### 1.5.1.2 Subgrupo 2 - Usuarios

Dentro de este subgrupo se encuentran personas que puedan estar dentro del demográfico objetivo de usuarios finales, entendemos que estos usuarios tienen menor tolerancia a ver mas allá de partes incompletas del producto por lo que solo tendrán acceso a *milestone builds* [g4], se hará un esfuerzo de ocultar o bloquear partes del producto que no estén funcionales. Este subgrupo es potencialmente mas grande por lo que tendrá como método de devolución un formulario estandarizado.

## 1.5.2 Herramientas y Tecnologías

### 1.5.2.1 Game Engines, Lenguajes, Editores

#### Unity

Optamos por elegir esta herramienta/motor de desarrollo [g5] de videojuegos debido a que se adapta perfectamente a las necesidades de nuestro proyecto. A su vez, nos permite desarrollar en C#, que es un lenguaje familiar para todos los integrantes del grupo. Por último, tenemos en cuenta la amplia comunidad que hay detrás de este

software, que nos facilitará el acceso a información o ayuda que necesitemos en el transcurso.

### Unreal Engine

Fue uno de los motores de videojuegos investigados para llevar a cabo este proyecto. Decidimos descartarlo, ya que es un motor diseñado para juegos muy demandantes en cuanto a gráficos, polígonos, simulaciones físicas e iluminación, lo cual no se ajusta a nuestras necesidades. Es un motor muy potente y pesado, si bien nos permite realizar todo lo que proponemos, preferimos contar con un motor mas liviano sin tanta funcionalidad que no vamos a utilizar.

### C#

Es una elección bastante natural debido a la familiaridad de los integrantes y su buena relación con el motor de videojuegos Unity. Tomando en cuenta la naturaleza de nuestro proyecto, nos parece apropiado utilizar un lenguaje simple y orientado a objetos, donde la relación entre estos últimos es de vital importancia.

### Unity Netcode y/o Mirror Networking

Son las dos librerías que elegimos como candidatas a utilizar. Unity Netcode es la librería de mediano nivel de networking de Unity, motivo principal de nuestra elección. Nos permitiría desarrollar el videojuego con su funcionalidad online pvp, sin tener que dedicarle recursos a protocolos de bajo nivel y *frameworks de networking*. Mientras que *Mirror Networking* es otra gran opción como librería de *networking* de alto nivel. Es un recurso gratis que se encuentra en la *store* de *Unity*. Dejamos planteadas estas dos tecnologías con el fin de tener una instancia posterior donde investigar y determinar una de las dos.

### MySql

Utilizaremos este sistema de gestión de base de datos relacional debido a que es conocido por su buena *performance*, algo que consideramos es importante.

### Php

Será el lenguaje que utilizaremos para realizar el acceso al servidor y base de datos, lo cual nos brindará una fuerte capa de seguridad.

## PhpMyAdmin

Será nuestro administrador de base de datos. Es simple y tiene una interfaz muy amistosa. La elección de MySQL para la base de datos y php como lenguaje de acceso a la misma, convierten a ésta en la herramienta ideal para este caso.

## Visual Studio Code / Microsoft Visual Studio Comunnity 2022

Ambas opciones nos permiten desarrollar en los lenguajes que elegimos. Uno ya cuenta con facilidades de integración con Unity y el otro permite la instalación de extensiones muy prácticas.

### 1.5.2.2 Control de Versión

#### Git

Es el sistema de control de versiones gratis que elegimos. Permite mantenernos actualizados en lo que al desarrollo de los demás respecta, y así ir construyendo el producto colaborativamente.

#### GitHub

Es el servicio en la nube que alojara nuestro sistema de control de versiones. Es altamente compatible con git, ya que fue desarrollado para facilitar su uso.

### 1.5.2.3 Comunicación

#### Discord

Es una herramienta gratis de comunicación muy versátil, con la cual los tres integrantes somos familiares. Nos permite la creación de un servidor propio, dónde podemos estar conectados constantemente. En este servidor podemos crear canales de distintas temáticas, lo cual nos brinda mayor organización. Allí podemos chatear, hablar con o sin vídeo, compartir pantalla.

#### Microsoft Teams

Principalmente, con esta herramienta estamos comunicados con el tutor del proyecto Santiago Fagnoni. Realizaremos las reuniones semanales pertinentes y consultas que tengamos a lo largo del proyecto.

## Google Jamboard

Es nuestro pizarrón digital, que nos permite trabajar de forma colaborativa. Allí podemos plasmar nuestras ideas, diseños y borradores en tiempo real.

### 1.5.2.4 Documentación

#### Jira

Es la herramienta elegida para la gestión del proyecto, algunos de los integrantes ya cuentan con experiencia laboral utilizando la misma. Cumple con las necesidades del equipo, permitiendo tener un backlog con tareas para ser asignadas a los distintos integrantes y realizar un correcto seguimiento del progreso del trabajo.

#### Confluence

Es una herramienta de colaboración con buena integración con Jira, allí pondremos todos los documentos pertinentes a este proyecto, incluso el que estamos redactando. Actúa como una suerte de wiki donde podremos establecer protocolos, guías y demás.

#### Google Drive

Nos brinda la posibilidad de almacenamiento gratis en la nube y nos da la facilidad de crear, compartir y modificar archivos simultáneamente.

#### Draw.io

Herramienta a utilizar para realizar los distintos diagramas. Permite trabajar en simultáneo y guardar automáticamente en git hub, lo que lo hace una herramienta flexible que se adecúa a nuestro control de versiones.

### 1.5.3 Estrategia de Control de Calidad

#### 1.5.3.1 Estándar de código:

Para facilitar trabajo en equipo, legibilidad y capacidad de detectar inconsistencias, nos adherimos a un estándar de código que se define a continuación, desviaciones de este estándar deben ser marcadas como errores a la hora de hacer un pull request de una funcionalidad con la intención de minimizar contribuciones que deban ser limpiadas a posteriori. Excepciones a seguir el estándar deben estar justificadas.

## Codificar en Inglés:

Todo código, comentario, variable, función, método, objeto, clase, componente y cualquier texto funcional deberá ser escrito en inglés. La única excepción a esta regla será para textos de visibles por el usuario que pueden ser en español, inglés o textos localizables.

✗ - Incorrecto:

```
public class jugador
{
    // Un metodo que hace una cosa
    public void unMetodo()
    {
    }
}
```

✓ - Correcto:

```
public class Player
{
    // A method that does a thing
    public void SomeMethod()
    {
    }
}
```

## Clases, Métodos y Namespaces en PascalCase

✗ - Incorrecto:

```
namespace game.objects.cards
{
    public class card
    {
        public void someCardMethod()
        {
        }
    }
}
```

✓ - Correcto:

```

namespace Game.Objects.Cards
{
    public class Card
    {
        public void SomeCardMethod()
        {
        }
    }
}

```

Cada llave en su propia línea, siempre usar llaves por mas que puedan ser omitidas.

✗ - Incorrecto:

```

public class Creator
{
    public void CreateThings(){
        for(int i=0; i<10; i++)
            CreateAThing();
    }
}

```

✓ - Correcto:

```

public class Creator
{
    public void CreateThings()
    {
        for(int i=0; i<10; i++)
        {
            CreateAThing();
        }
    }
}

```

Atributos:

Atributos auto implementados son preferibles y pueden declararse con sus llaves en una sola linea. Siempre usar modificadores de nivel de acceso.

✗ - Incorrecto:

```

public class Player
{
    string myName;
    int myAge;
    public string myNameExternal
    {
        get
        {
            return this.myName;
        }
        set
        {
            this.myName= value;
        }
    }
}

```

✓ - Correcto:

```

public class Player
{
    private int myAge;
    public string myName { get; set; }
}

```

Evitar siglas:

Evitar nombres compuestos de siglas o acrónimos, es preferible nombres descriptivos que eviten depender de comentarios.

✗ - Incorrecto:

```

public class TurnManager
{
    // Checks the state of the current turn to see if
    // an active unit has reached a win condition
    public void EvaluateAUWinCondition()
    {
    }
}

```

✓ - Correcto:

```

public class TurnManager
{
}

```

```

public void EvaluateActiveUnitsWinCondition()
{
}
}

```

### Atributos, variables y scope:

Usamos camel case para las variables. Todos los atributos de una clase deben ser adornados con el prefijo “my”, los parámetros que se reciben en una función deben ser adornados con el prefijo “a”, variables de periodos cortos de vida no deben ser adornadas con “a” o “my”. Las listas y arreglos deben ser nombrados en plural.

✓ - Correcto:

```

public class UnitManager
{
    private List<Unit> myUnits;

    public void SetUpUnits(bool aIsInitialSetup, int aNumberOfUnits)
    {
        if(aIsInitialSetup)
        {
            myUnits = new List<Unit>();
        }

        for(int i=0; i<aNumberOfUnits; i++)
        {
            string logText = "Setting up unit number: " + i;
            Debug.Log(logText);

            if(aIsInitialSetup)
            {
                Unit unit = new Unit(i);
                unit.Init();
                myUnits.add(unit);
            }
            else
            {
                myUnits(i).Init();
            }
        }
    }
}

```

Métodos delegados deben ser marcados con sufijo “Callback”:

✓ - Correcto:

```
// Declare a delegate type for processing a user:  
public delegate void ProcessUserCallback(User aUser);
```

Eventos y acciones reactivas a inputs [g7] de usuario deben iniciar con el prefijo “On”

✓ - Correcto:

```
public class TurnManager  
{  
    public void OnTurnPaused()  
    {  
    }  
}
```

Booleanos:

Atributos booleanos y métodos que resuelven tipos de dato booleanos deben tener prefijo Is, Has, Will

✓ - Correcto:

```
public class Card  
{  
    public bool myIsJoker = false;  
    public bool myIsRed = false;  
  
    public bool IsRedJoker()  
    {  
        return myIsJoker && myIsRed;  
    }  
}
```

#### 1.5.4 Método de trabajo

La metodología que se optó para el proyecto es una adaptación de una metodología ágil de desarrollo ‘Scrum’. Se considera que esta metodología se adapta de buena manera a nuestros objetivos y necesidades.

Las metodologías ágiles, según Sommerville [1], prescriben que los desarrolladores debe comenzar en los requerimientos con mayor importancia para el usuario y el correcto funcionamiento del producto final. Esto es importante para un videojuego,

como el que se desarrollará, ya que tener versiones funcionales para mostrar a grupos de foco es un punto clave del proceso de desarrollo.

Una característica clave de la metodología ‘Scrum’ es la división del período de trabajo en sub-períodos llamados ‘sprints’, que en este proyecto serán de dos semanas. Estos sprints determinan las tareas a realizar por cada integrante, y una vez cumplido el tiempo de cada sprint, el grupo de desarrollo revisa y comenta el trabajo realizado en el mismo sprint. Estas reuniones permiten al grupo adaptar los siguientes sprints dependiendo del progreso realizado.

Este método de trabajo permite al grupo a adaptarse de manera flexible y consciente a cambios o atrasos que puedan surgir en el proceso de desarrollo. Esto es importante ya que por el tiempo limitado que existe para este proyecto, es posible que los requerimientos cambien durante su transcurso.

#### 1.5.4.1 Incrementos e Iteraciones

Como ya fue mencionado, los sprints serán de dos semanas. Antes de cada sprint, se realizará una reunión de planificación donde el grupo discutirá y definirá los objetivos del mismo. Se generará una pila de tareas (backlog) a realizarse para este sprint. Después de cada sprint, habrá otra reunión de índole retrospectiva donde se verificará el trabajo realizado, qué no se pudo realizar y qué mejoras se podrán hacer para acoplarse al cronograma del proyecto.

En varios puntos del cronograma, se planea tener versiones entregables del proyecto, que serán mostradas a los grupo de foco mencionados anteriormente. El equipo usará la reunión inicial del sprint correspondiente para discutir e interiorizar el *feedback* [g6] recibido para desarrollar un mejor producto.

#### 1.5.5 Cronograma

Contamos 129 días totales desde el inicio del desarrollo hasta la fecha de entrega final. De estos días 36 son fines de semana y 93 son días laborables, para este proyecto consideramos que todos los días son laborables para permitir que cada desarrollador maneje sus horarios de la manera que le sea más conveniente.

##### 1.5.5.1 Fechas Clave

- **Inicio:** Lunes 16 de Mayo, primer día de desarrollo del primer sprint.
- **Deadline\_01:** Jueves 16 de Junio, primer informe de avance.
- **Deadline\_02:** Jueves 28 de Julio, segundo informe de avance.
- **Deadline\_03:** Jueves 22 de Setiembre, entrega final.

Rango de días	Fecha A	Fecha B
---------------	---------	---------

32	<b>Inicio</b>	<b>Deadline_01</b>
42	<b>Deadline_01</b>	<b>Deadline_02</b>
56	<b>Deadline_02</b>	<b>Deadline_03</b>

Tabla 5. Representación de fechas clave

### 1.5.5.2 Sprints y Épicas

Decidimos hacer coincidir las épicas de desarrollo con las fechas claves señaladas. Dividimos el tiempo disponible en los siguientes sprints de 14 días cada uno aproximadamente.

SprintId	Inicio	Fin	Épica
<b>Sprint_01</b>	16 May 2022	29 May 2022	<b>Infrasturctrure &amp; Design</b>
<b>Sprint_02</b>	30 May 2022	22 Sep 2022	<b>Infrasturctrure &amp; Design</b>
<b>Sprint_03</b>	17 Jun 2022	30 Jun 2022	<b>Game Development</b>
<b>Sprint_04</b>	01 Jul 2022	14 Jul 2022	<b>Game Development</b>
<b>Sprint_05</b>	15 Jul 2022	28 Jul 2022	<b>Game Development</b>
<b>Sprint_06</b>	29 Jul 2022	11 Aug 2022	<b>Polish &amp; Testing</b>
<b>Sprint_07</b>	12 Aug 2022	25 Aug 2022	<b>Polish &amp; Testing</b>
<b>Sprint_08</b>	26 Aug 2022	08 Sep 2022	<b>Polish &amp; Testing</b>
<b>Sprint_09</b>	09 Sep 2022	22 Sep 2022	<b>Polish &amp; Testing</b>

Tabla 6. Mapeo entre sprints y épicas

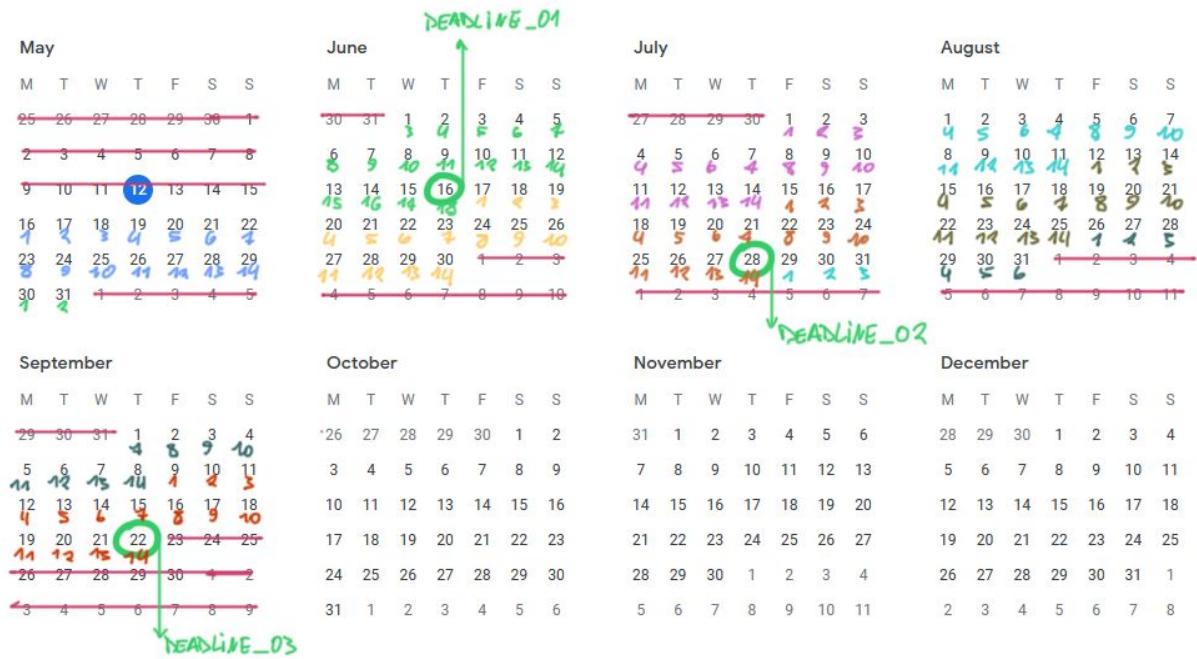


Figura 1. Calendario de proyecto

## 1.6 Requerimientos Funcionales

Los requerimientos listados representan funcionalidades (stories) que se desglosarán en tareas (tasks) para popular el backlog(pila de tareas) de todo el proyecto.

Id	Nombre	Descripción	Prioridad

RF-1	S01_Login	<p>Pantalla con formulario de login y botón de navegación a pantalla de registro</p> <p><b>Login:</b></p> <ul style="list-style-type: none"> <li>• Título de formulario “Login”</li> <li>• Input de texto para nombre de usuario mínimo 8 caracteres y máximo 32 caracteres</li> <li>• Input de texto para email con validación de formato</li> <li>• Input para contraseña con validación de formato, mínimo 8 caracteres y máximo 32 caracteres</li> <li>• Botón para ingresar</li> </ul> <p><b>Navegación:</b></p> <ul style="list-style-type: none"> <li>• Botón a formulario de registro</li> </ul>	Muy alta
RF-2	S02_Register	<p>Pantalla con formulario de registro y botón de navegación a pantalla de login</p> <p><b>Registro:</b></p> <ul style="list-style-type: none"> <li>• Título de formulario “Register”</li> <li>• Input de texto para nombre de usuario mínimo 8 caracteres y máximo 32 caracteres</li> <li>• Input de texto para email con validación de formato</li> <li>• Input para contraseña con validación de formato, mínimo 8 caracteres y máximo 32 caracteres</li> <li>• Input para repetir contraseña con validación de formato, debe ser igual a la ingresada previamente, mínimo 8 caracteres y máximo 32 caracteres</li> </ul> <p><b>Navegación:</b></p> <ul style="list-style-type: none"> <li>• Botón a formulario de login</li> </ul>	Muy alta

RF-3	S11_AccountRecovery	<p>Pantalla de recuperación de contraseña con navegación a pantalla inicial (Login)</p> <p>Formulario:</p> <ul style="list-style-type: none"> <li>• Input de texto para email con validación de formato</li> <li>• Botón de “Solicitar nueva contraseña”</li> </ul> <p>Navegación:</p> <ul style="list-style-type: none"> <li>• Botón a formulario de login</li> </ul>	Muy baja
RF-4	S03_MainMenu	<p>Pantalla principal con botones de navegación</p> <p>Botones:</p> <ul style="list-style-type: none"> <li>• Play</li> <li>• Decks</li> <li>• Settings</li> <li>• Log out</li> </ul>	Muy alta
RF-5	S05_PlayMenu	<p>Pantalla de navegación al tipo de partida con botones:</p> <ul style="list-style-type: none"> <li>• Public match (Busca partida pública dirigiéndose a S08_PublicMatchmaking)</li> <li>• Private match (Navega a S06_SetPrivateMatch)</li> <li>• Back (Navega a pantalla previa S03_MainMenu)</li> </ul>	Muy alta

RF_6	S04_Decks	<p>Pantalla de administración de decks (deckbuilding). Debe contar con un listado de los decks seleccionables pertenecientes al usuario con un botón de creación de deck. Deben existir botones con acciones:</p> <ul style="list-style-type: none"> <li>• Edit</li> <li>• Save</li> <li>• Cancel</li> <li>• Back (Navega a pantalla previa S03_MainMenu)</li> </ul> <p>También debe haber un panel que muestre las cartas y permita seleccionar la cantidad de copias de una carta que tendrá el mazo seleccionado.</p> <p>Debe haber un contador con la cantidad de cartas seleccionadas hasta el momento.</p>	Alta
RF_7	S06_SetPrivateMatch	<p>Pantalla de partida privada con botones que permiten crear una partida privada o unirse a una ya existente mediante un input con id de partida.</p> <p>Botones e inputs:</p> <ul style="list-style-type: none"> <li>• Host private match</li> <li>• Input private match id</li> <li>• Join private match</li> </ul>	Media

RF_8	S07_PrivateMatchLobby	Pantalla pre partida donde se muestra el identificador de la misma y los jugadores ingresados en el <i>lobby</i> [g8]. Se debe permitir la selección de mazo en un combo desplegable. Debe haber un botón o checkbox que permita marcarse como pronto para jugar (Ready). Debe haber un botón que permita salir del <i>lobby</i> (Leave). Se debe mostrar a modo informativo el estado del otro jugador (Ready/Unready) con un checkbox.	Media
RF_9	S08_PublicMatchmaking	Pantalla con título “Matchmaking...” que muestre que se esta buscando activamente una partida mediante una animación de carga. Debe tener un botón de cancelar la búsqueda de partida (Cancel)	Baja
RF_10	S09_PublicMatchLobby	Pantalla pre partida donde se muestra un timer con el tiempo restante para aprontarse y los jugadores ingresados en el <i>lobby</i> . Se debe permitir la selección de mazo en un combo desplegable. Debe haber un botón o checkbox que permita marcarse como pronto para jugar (Ready). Debe haber un botón que permita salir del <i>lobby</i> (Leave). Se debe mostrar a modo informativo el estado del otro jugador (Ready/Unready) con un checkbox. Una vez finaliza el tiempo o ambos jugadores están prontos (Ready), se debe proceder a la pantalla S10_MatchReady	Alta

RF_1 1	S10_MatchReady	Pantalla pre partida donde se muestra un timer con los segundos restantes para que comience la partida. Se debe mostrar la información y estado de los jugadores que se encuentran en el <i>lobby</i> .	Baja
RF_1 2	S12_Match	Pantalla donde se desarrolla una partida	Muy alta
RF_1 3	D01_LoginServer	Deploy de servidor de login que se encarga de manejar el ingreso y registro de usuarios al sistema y otorgar tokens de sesión.	Muy alta
RF_1 4	D02_Database	Deploy de base de datos en el Cloud Hosting Service.	Muy alta
RF_1 5	D03_GameServer	Deploy del game server que administra las partidas, controla el flujo de las partidas e instancia lobbies.	Muy alta
RF_1 6	S13_EndMatch	Pantalla post partida que muestra el nombre del jugador ganador.	Muy baja
RF_1 7	S14_Settings	Pantalla de configuración del cliente del juego.	Baja

Tabla 7. Requerimientos funcionales

## 1.7 Requerimientos No Funcionales

Los requerimientos listados representan características que el producto final debe cumplir para alcanzar el estándar de calidad deseado pero que no ocurren como resultado de tareas específicas.

<b>Id</b>	<b>Descripción</b>
RNF_1	El cliente debe poder ejecutar el juego en sistema operativo Windows.
RNF_2	El sistema debe estar disponible en todo momento.
RNF_3	El código de todo el sistema debe estar en idioma Inglés.
RNF_4	Manual de usuario que detalle como instalar/utilizar el juego.

RNF_5	Todo actualización del juego debe ser realizada entre las 4 AM y 7 AM (UTC -03:00).
RNF_6	El idioma de la interfaz de usuario debe estar en un idioma unificado y consistente.
RNF_7	Comentar métodos con el fin de facilitar el entendimiento del código.
RNF_8	Validar que todos los datos a ingresar a la base de datos sean del tipo de dato correcto.

Tabla 8. Requerimientos no funcionales

## 1.8 Pre-Producción

### 1.8.1 Stack de Tecnologías

#### 1.8.1.1 Infraestructura

La infraestructura del proyecto tiene dos grandes componentes, servicios hosteados en la nube y el cliente ejecutable compilado con Unity. Este diagrama estará versionado y puede evolucionar a lo largo del desarrollo del proyecto.

#### Servicios

##### DataBase

Contiene información de cada usuario, credenciales de acceso y stats de juego.

##### LoginServer

Verifica credenciales y está encargado de manejar el ingreso de usuarios al sistema. Entrega tokens de sesión tanto al GameServer como a cada usuario que se conecta para identificar a esos usuarios en el GameServer.

##### GameServer

Administra todas las partidas del juego que pueden existir al mismo tiempo. Instancia lobbies y se encarga de llenarlos. Controla el flujo de las partidas en caso de ser necesario.

## Clients

### GameClient

Cada usuario cuenta con un ejecutable compilado con Unity para Windows que se conecta con los servicios y otros clientes para usar sistema.

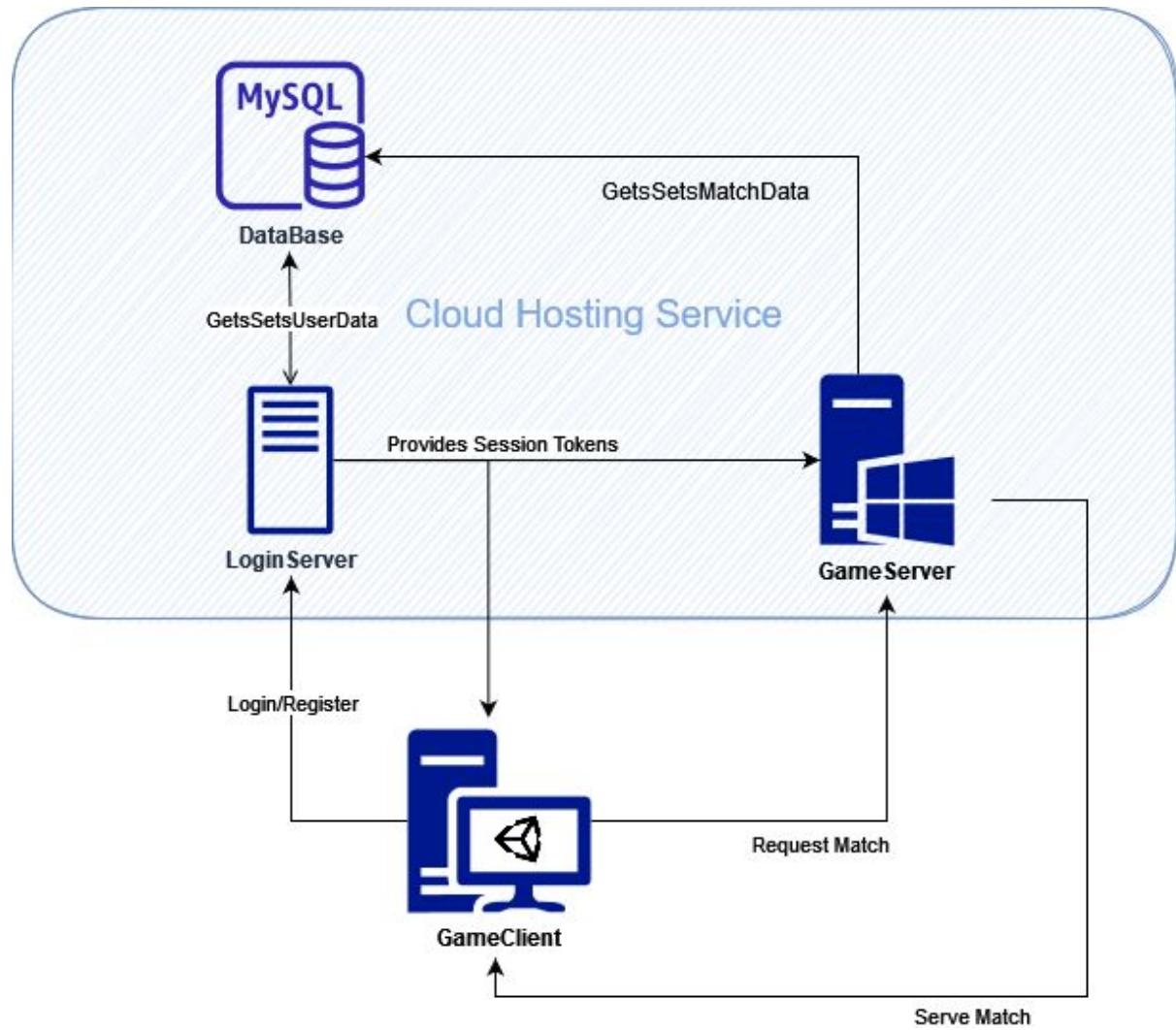


Figura 2. Stack de tecnologías v01

### 1.8.2 Diagramas de Clases

Representación de las entidades que componen la lógica de negocio, estos diagramas estarán versionados y pueden evolucionar a lo largo del desarrollo del proyecto.

### 1.8.2.1 Administración de Juegos

GameInstanceManager crea Lobby(s) que hasta que se llenan con User(s), una vez que ambos aceptan el encuentro se crea una nueva instancia de Game, con Player(s) instanciados a partir de los User(s) y se destruye el Lobby.

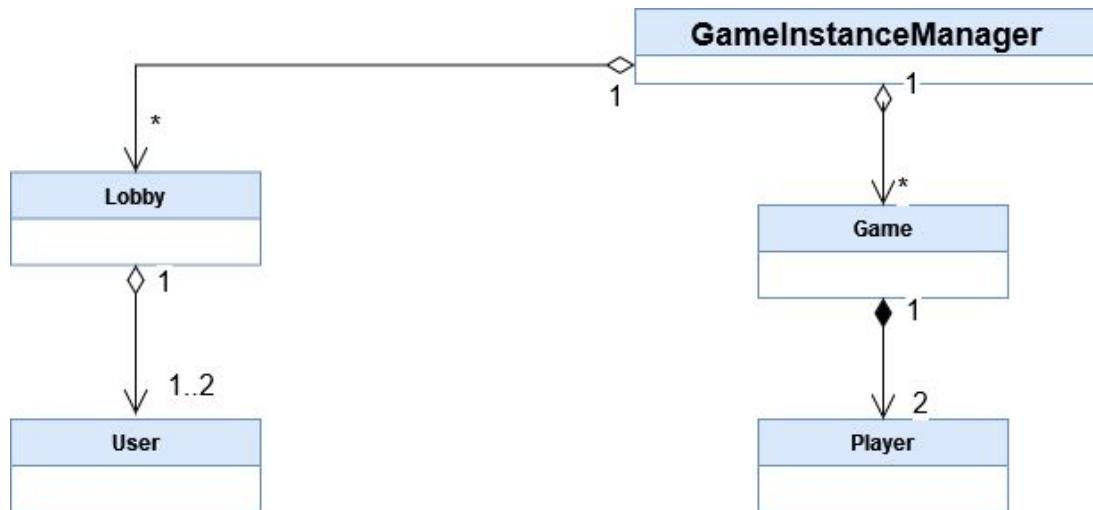


Figura 3. Administración de juegos v01

### 1.8.2.2 Juegos

Cada Game está compuesto por Player(s) que deben transcurrir una cantidad indeterminada de Turn(s) hasta que se alcanza una condición de victoria para uno de ellos. Dentro de cada juego los Player(s) tienen acceso a una Deck, esta Deck es una combinación de Card(s) que Player ha elegido, se define previamente al Game y puede persistir a lo largo de varios Game(s).

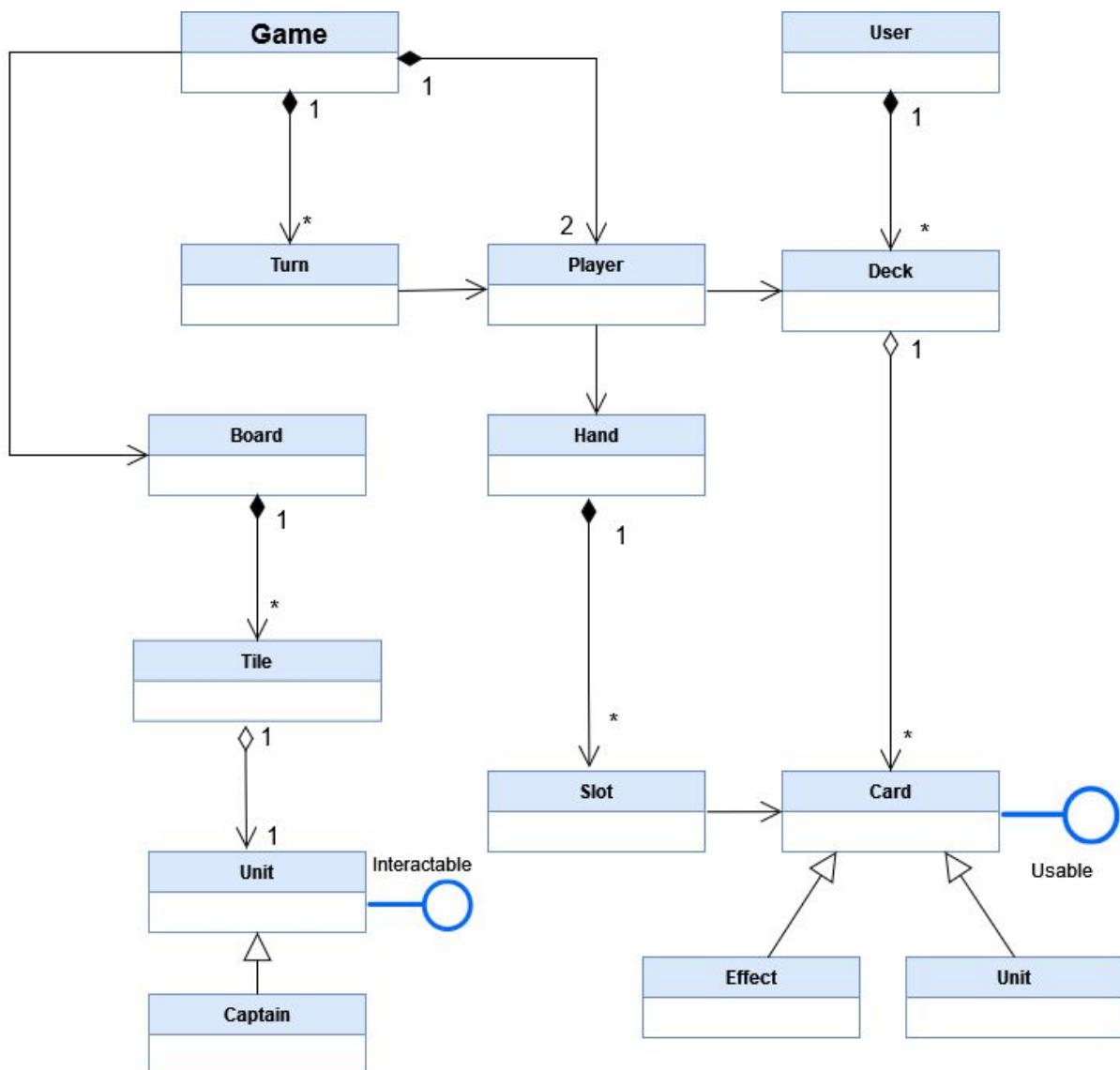


Figura 4. Juegos v01

### 1.8.3 Flujo de Pantallas

Flujo de pantallas para toda la aplicación del cliente, este diagrama estará versionado y puede evolucionar a lo largo del desarrollo del proyecto.

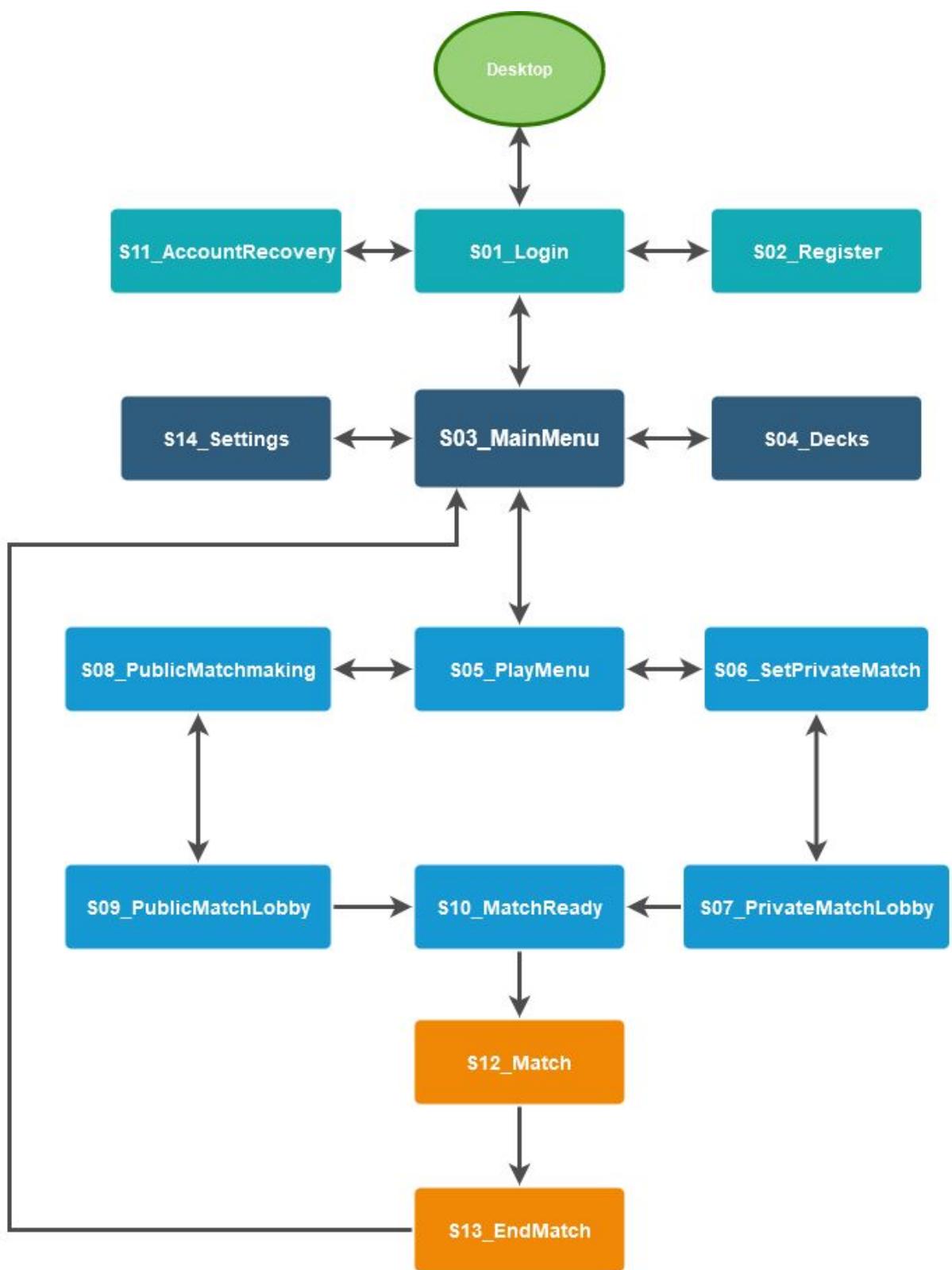


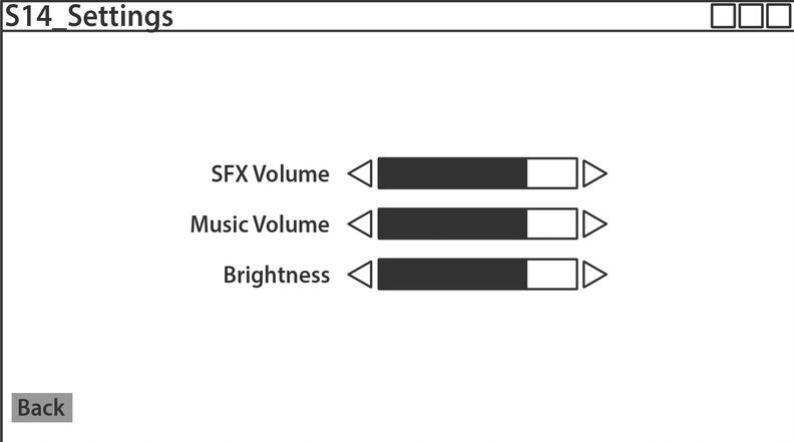
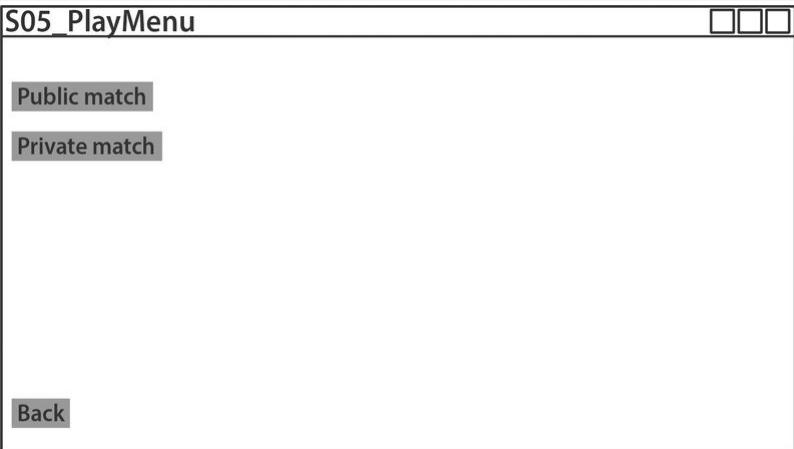
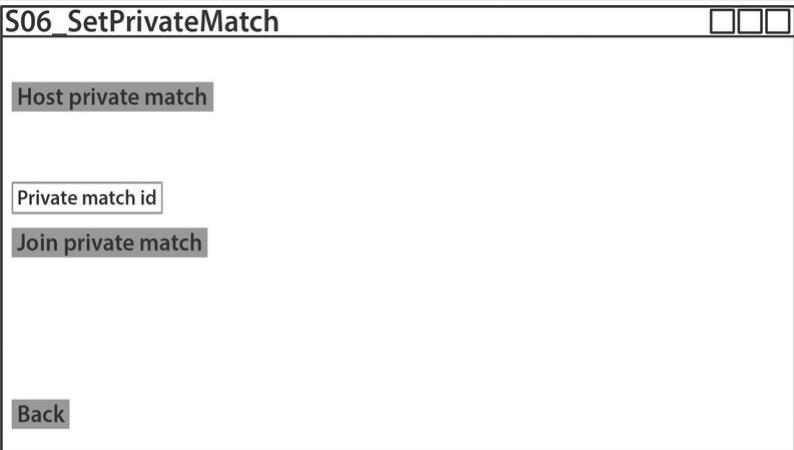
Figura 5. Flujo de pantallas v01

#### 1.8.4 Mockups de Pantallas

Representación preliminar de la composición grafica de las pantallas, estos diseños estarán versionados y evolucionaran a lo largo del desarrollo del proyecto.

Nombre	Versión	Mockup
S01_Login	1	<p>S01_Login</p> <p>The mockup shows a window titled "S01_Login". It contains a "Login" button, a "Register" button, a "Username" input field, a "Password" input field, a "Forgot password?" link, and a "Send" button. There is also an "Exit" button at the bottom left.</p>
S02_Register	1	<p>S02_Register</p> <p>The mockup shows a window titled "S02_Register". It contains a "Login" button, a "Register" button, an "Email" input field, a "Username" input field, a "Password" input field, a "Confirm password" input field, and a "Send" button. There is also an "Exit" button at the bottom left.</p>

S11_AccountRecovery	1	<p><b>S11_AccountRecovery</b></p> <p>This window contains fields for 'Email' and a 'Send' button.</p> <p>Buttons: Login, Register, Account recovery, Back.</p>
S03_MainMenu	1	<p><b>S03_MainMenu</b></p> <p>Buttons: Play, Decks, Settings, Log out.</p>
S04_Decks	1	<p><b>S04_Decks</b></p> <p>Left sidebar: MyDeck_1 (highlighted), MyDeck_2, New Deck ..., Edit, Save, Cancel, Back.</p> <p>Right area: A 3x5 grid of card slots. Some slots contain numbers (1, 2, 3, 0) and small arrows. A vertical bar on the right indicates 'Selected Cards: 7' and 'Max Cards: 10'.</p>

S14_Settings	1	<p><b>S14_Settings</b></p>  <p>SFX Volume</p> <p>Music Volume</p> <p>Brightness</p> <p><b>Back</b></p>
S05_PlayMenu	1	<p><b>S05_PlayMenu</b></p>  <p>Public match</p> <p>Private match</p> <p><b>Back</b></p>
S06_SetPrivateMatch	1	<p><b>S06_SetPrivateMatch</b></p>  <p>Host private match</p> <p>Private match id</p> <p>Join private match</p> <p><b>Back</b></p>

S07_PrivateMatchLobby	1	<p><b>S07_PrivateMatchLobby</b></p> <p>Private match id: XYZ</p>
S08_PublicMatchmaking	1	<p><b>S08_PublicMatchmaking</b></p> <p>Matchmaking...</p> <p>Cancel</p>
S09_PublicMatchLobby	1	<p><b>S09_PublicMatchLobby</b></p> <p>Lock In!</p> <p>13s</p>

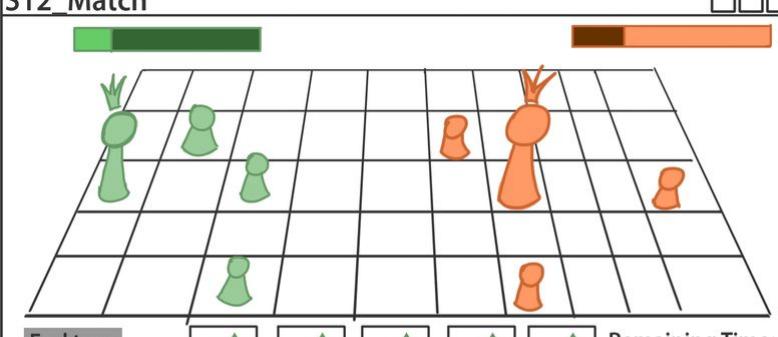
S10_MatchReady	1	<b>S10_MatchReady</b> Match starting in 5s My player name MyDeck_1 Their player name <input checked="" type="checkbox"/> <input checked="" type="checkbox"/>
S12_Match	1	<b>S12_Match</b> End turn  Remaining Time: 7s
S13_EndMatch	1	<b>S13_EndMatch</b> Victory! Exit

Tabla 9. Mockups de pantallas

### 1.8.5 Arte Conceptual de Gameplay

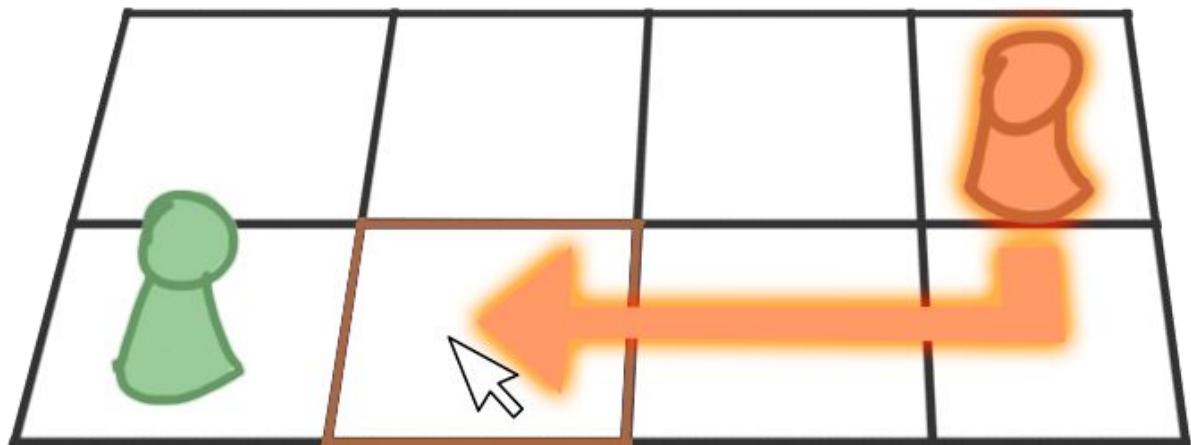


Figura 6. Movimiento de una unidad



Figura 7. Ataque cuerpo a cuerpo de una unidad

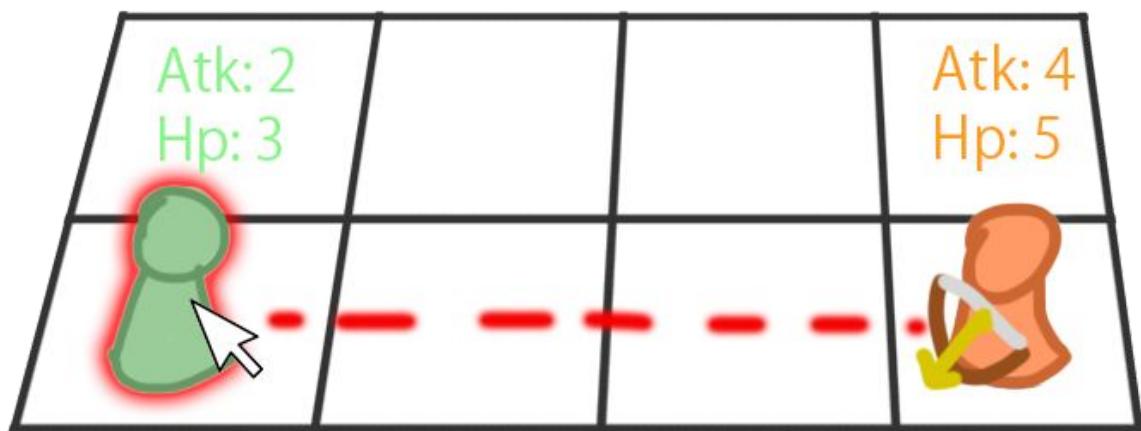


Figura 8. Ataque a distancia de una unidad

## 2 Proyecto

### 2.1 Sprint 1

#### 2.1.1 Resumen

Esta primera etapa fue planificada para abordar el tema infraestructura. Consideramos que éste es un tema crítico, que forma parte de los cimientos del proyecto y que, por lo tanto, era importante encararlo temprano.

Para llevarlo a cabo, hemos dedicado recursos a la investigación de tecnologías a utilizar. Analizamos los posibles proveedores de hosting de servidores para videojuegos, servicios que nos permitieran la autenticación y registro de usuarios, así como la persistencia de datos. Otro de los puntos a ser investigado eran las alternativas de herramientas para el manejo y control de versiones.

Luego, tuvimos reunión de puesta a punto sobre lo analizado con el fin de tomar una decisión para implementar el servidor y su relación con el cliente.

Por último, dedicamos un día acorde a lo planeado para reunirnos a comenzar a definir mecánicas del juego con el fin de tener una descripción más detallada del *gameplay*.

#### 2.1.2 Tareas

Tareas a llevar a cabo en este sprint.

<b>Id</b>	<b>Título</b>	<b>Tiempo Estimado/Real hs</b>	<b>Estado</b>
HOOD-57	Investigar Cloud Hosting Services	15/15	Finalizada
HOOD-52	Implementar repositorio de desarrollo	10/7	Finalizada
HOOD-51	Investigar administración de usuarios	40/34	Finalizada
HOOD-50	Investigar administración de juegos	40/34	Finalizada
HOOD-64	Configurar confluence para cumplir con 302	14/10	Finalizada

Tabla 10. Tareas en Sprint 1

Destinamos un promedio de 7hs semanales a reuniones en este sprint.

### 2.1.3 Investigación

Dispusimos de recursos para investigar las diferentes opciones que existen para el *hosting* de servidores para videojuegos, hacer prototipados y recabar fuentes de información para presentar en la reunión de puesta a punto. También dispusimos de recursos para investigar otros temas importantes como el manejo de usuarios y la herramienta de control de versión.

### 2.1.4 Investigación - Servicio de alojamiento de servidor

Concluimos que la mejor opción para nuestro proyecto era utilizar AWS y su servicio llamado “AWS Gamelift”. Éste servicio consiste en una solución de alojamiento de servidores que puede implementar, operar y escalar servidores dedicados y de bajo costo en la nube para juegos multijugador basados en sesiones.

### 2.1.5 Investigación - Servicio de manejo de usuarios

En base al hallazgo del servicio de AWS Gamelift, decidimos seguir por la misma línea utilizando otro servicio de Amazon, lo cual nos garantizaba la compatibilidad de ambos. Es por esto que optamos por “AWS Cognito”, que nos ofrece autenticación, autorización y administración de usuarios. Sus dos componentes principales son los grupos de usuarios y los grupos de identidades. Uno se encarga de proporcionar directorios de usuarios con opciones para registrarse e iniciar sesión. El otro nos permite conceder a los usuarios el acceso a otros servicios de AWS.

### 2.1.6 Investigación - Herramienta de control de versión

Otro de los temas a investigar era la tecnología para llevar el control de versiones. Si bien en el anteproyecto, habíamos tomado una decisión, sabíamos que podrían existir alternativas que nos faciliten esta tarea. Éste fue el momento donde “Plastic SCM” tomó un rol importante para el proyecto. A diferencia de GitHub, herramienta establecida en el anteproyecto, consideramos que “Plastic SCM” es una herramienta mucho más práctica, que se adapta de mejor forma a nuestro proyecto. Ésta fue desarrollada especialmente para Unity y nos proporciona una interfaz agradable a modo de plugin acoplado al IDE.

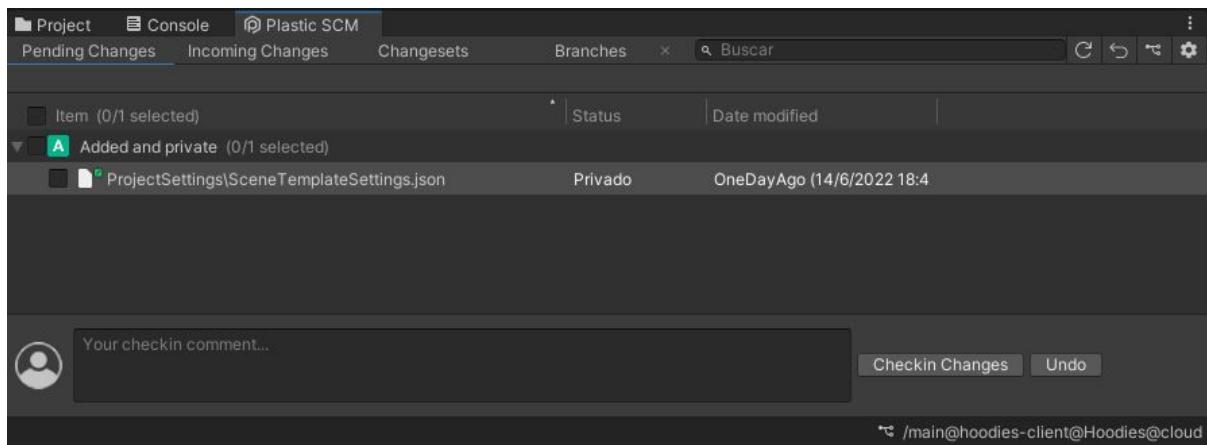


Figura 9. Plugin de PlasticSCM en Unity con cambios pendientes

## 2.1.7 Diseño - Descripción del Gameplay

El juego consiste de dos componentes claves: deckbuilding y combate.

### 2.1.7.1 Deckbuilding

Un jugador tendrá una lista de cartas asociada a su cuenta que podrá utilizar para crear o modificar mazos (deckbuilding).

### 2.1.7.2 Mazo

Cada mazo tendrá un límite de 50 cartas, y cada jugador tendrá un límite de 25 mazos únicos. Cada mazo podrá tener varias copias de una misma carta, pero su cantidad se verá limitada por la cantidad de copias que tiene el jugador en su lista de cartas y por un límite de ejemplares por mazo que es determinado por cada tipo de carta.

### 2.1.7.3 Cartas

Las cartas están divididas en tres tipos: Nave Nodriza, Unidad y Tecnología. Todas las cartas salvo la Nave Nodriza tienen asociado un costo medido en “energía”.

#### Unidad

Cada unidad tiene asociada a sí valores que determinan su comportamiento en la fase de combate: Velocidad, Integridad Estructural, Rango de Ataque y Poder. Las funciones de estos valores serán explicados más abajo en la sección de combate. Además de estos valores, que son universales entre las unidades, una unidad podrá tener una habilidad especial, que modifica su comportamiento o le permite activar algún efecto en la fase de combate.

## Nave Nodriza

La Nave Nodriza es un tipo especial de unidad sin costo que debe estar presente en todo mazo. Además, sólo podrá haber una de ellas por mazo. Todas las Naves Nodrizas tienen una habilidad especial. Si la Nave Nodriza es destruida, el jugador dueño de la nave pierde el juego.

## Hechizo

Las cartas de tecnología funcionan como las habilidades especiales de las cartas, pero éstas son descartadas unas vez que se activa su habilidad.

### 2.1.7.4 Combate

En la fase de combate, cada jugador elegirá uno sus mazos que creó en la fase de deckbuilding para luego enfrentarse en un campo de batalla. La acción se desarrollará en turnos en un orden determinado al comenzar la partida, que se repite hasta que ésta finaliza. Cada turno tiene un límite de tiempo, si este tiempo se agota, el turno automáticamente pasará al jugador siguiente que corresponda. Cada jugador también tiene la habilidad de terminar su turno en cualquier momento presionando el botón “End Turn”.

### 2.1.7.5 Campo de batalla

Esta fase del juego se desarrollará en un tablero rectangular dividido en cuadrantes. Estos cuadrantes podrán tener propiedades especiales que afecten a las unidades o a los hechizos.

### 2.1.7.6 Fase inicial

Al comenzar el juego, cada mazo se mezclará y se repartirán 5 cartas a la mano de cada jugador de sus respectivos mazos. El juego entonces, determinará aleatoriamente a quién le corresponde el primer turno y colocará a la Nave Nodriza de cada jugador en posiciones opuestas predefinidas en el tablero.

### 2.1.7.7 Fase de Ataque

Una vez posicionadas todas las Naves Nodrizas, comienza la fase de ataque.

Al iniciar su turno, y en todos sus turnos subsecuentes los jugadores recibirán un punto extra de energía, hasta un máximo de 10. También agregarán una carta nueva a su mano en caso de tener menos de 5 cartas hasta agotar el mazo.

Acciones en un turno:

- Posicionar una unidad en el tablero
- Activar una carta de tecnología
- Activar la habilidad especial de una unidad

Cada una de las acciones anteriores tiene un costo de energía. Al realizar la acción, su costo se restará al total de energía que tiene disponible el jugador. Si una acción cuesta más que el total disponible de un jugador, no se podrá hacer.

Los jugadores también podrán utilizar sus unidades de varias formas. Algo que se explicará en la siguiente sección:

#### 2.1.7.8 Comportamiento de unidades

Salvo en casos especiales, al posicionar una unidad, ésta permanecerá inactiva hasta el siguiente turno del jugador que la invocó. Una unidad inactiva no podrá realizar ningún tipo de acción.

Una vez que la unidad está activa ésta podrá realizar las siguientes acciones:

- Mover
  - Cada unidad tiene un valor “Velocidad” que representa la distancia en cuadrantes que se puede mover en un turno dado. Por ejemplo, una unidad con “Velocidad” 3 se podrá mover 3 cuadrantes en un turno. Una vez que una unidad se movió, esta no podrá moverse de nuevo hasta el siguiente turno.
- Atacar
  - Una unidad podrá atacar a otra unidad enemigo siempre y cuando esta esté en su “Rango” y no tenga otras unidades entre el atacado y el atacante. Por ejemplo, una unidad con “Rango” 2 podrá atacar a otra siempre y cuando se encuentre a 2 cuadrantes de distancia y no tenga a otra unidad entre ellas. Una vez que una unidad atacó, esta pasará a estar “inactiva”.
  - La eficacia de un ataque es determinada por el “Poder” del atacante y la “Integridad Estructural” del atacado. El valor del “Poder” se le resta a la “Integridad Estructural” del atacado, y si esta última es menor o igual a 0, la unidad es destruida.
- Habilidad Especial
  - Ciertas unidades tendrán la posibilidad de activar una habilidad especial. Los efectos de esta habilidad variarán por cada habilidad y serán descriptos dentro del juego.
  - Activar habilidades especiales puede tener un costo de energía asociado.

Si una unidad no es capaz de realizar ninguna acción en ese turno, automáticamente pasará a un estado inactivo.

## 2.1.8 Implementación - Control de Versión

En esta primer instancia la única tarea de implementación era la correspondiente a la creación del repositorio del proyecto con la herramienta Plastic SCM. Dividimos el proyecto en dos sub-proyectos, uno que contiene el código dedicado al servidor y otro que contiene el código dedicado al cliente.

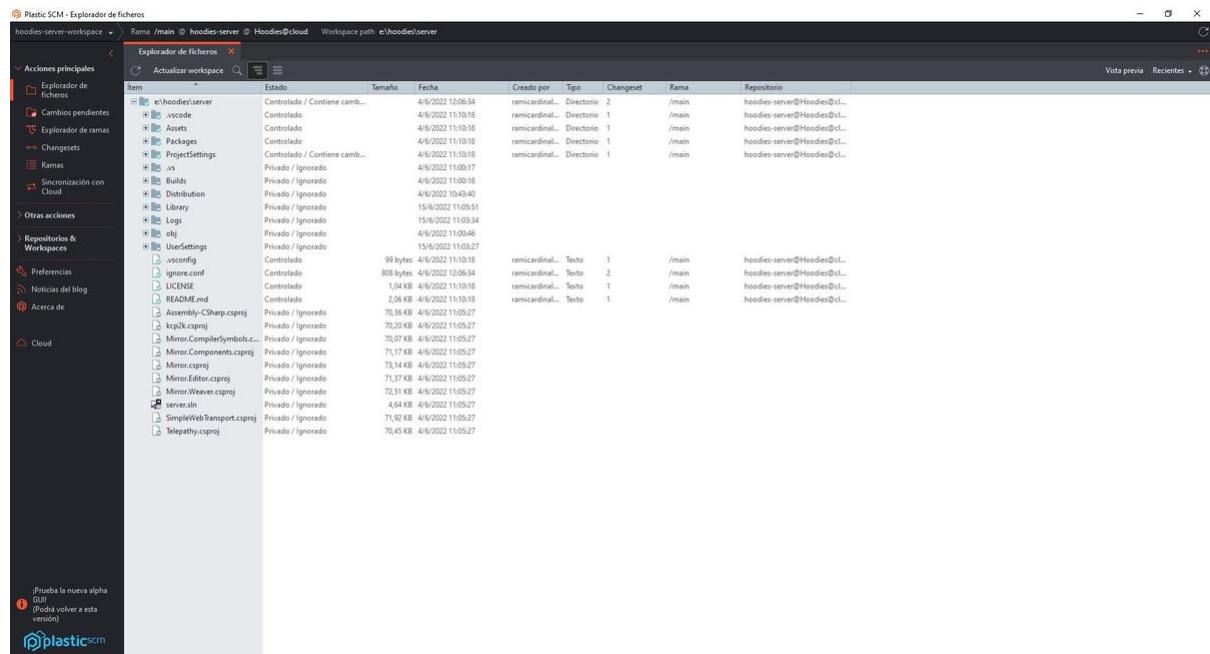


Figura 10. Aplicación de escritorio PlasticSCM con repositorio del código de servidor

## 2.2 Sprint 2

### 2.2.1 Resumen

En esta segunda etapa teníamos planificado continuar trabajando en el tema infraestructura pero con enfoque en la implementación. Sabíamos que ésta podía ser muy compleja y debía tener un cierre de cumplimiento. Es por esto que consideramos apropiado definir una meta inicial básica para cada una de estas tareas. Por otro lado, en la fase de investigación y análisis realizada en el sprint anterior habíamos optado por seguir por la línea de AWS, con el servicio “Cognito” que nos permite registrar usuarios y autenticarlos, así como también nos permite otorgar distintos permisos y privilegios. Por lo tanto el siguiente objetivo planteado para este sprint fue lograr un registro e inicio de sesión funcional, que pueda ser integrado al proyecto.

## 2.2.2 Tareas

Tareas a llevar a cabo en este sprint.

<b>Id</b>	<b>Titulo</b>	<b>Tiempo Estimado/ Real hs</b>	<b>Estado</b>
HOOD-62	Build - Diseño e Infraestructura	25/14	Finalizada
HOOD-61	Implementar Matchmake	60/48	Finalizada
HOOD-60	Implementar Register-Login	40/42	Finalizada
HOOD-59	Investigar administración de datos	10/8	Finalizada

Tabla 11. Tareas Sprint 2

Destinamos un promedio de 5hs semanales a reuniones en este sprint.

## 2.2.3 Investigación

El principal objetivo de investigación en este sprint fue comunicación entre servicios de AWS y persistencia de datos de usuario.

### 2.2.4 Investigación - Comunicación entre servicios

Durante la investigación y a raíz de los servicios elegidos en el primer sprint, surgió la posibilidad de utilizar el servicio AWS Lambda [v13]. Es un servicio informático sin servidor, basado en eventos que nos permite ejecutar código para realizar acciones, acceder o modificar información de otros servicios AWS.

### 2.2.5 Investigación - Persistencia de Datos

Como muchos servicios *online*, tenemos la necesidad de mantener información de nuestros usuarios. En el capítulo [Herramientas y Tecnologías](#) mencionamos nuestra intención de usar [PhpMyAdmin](#) y [MySQL](#) como tecnologías para la persistencia de datos, pero avanzado el desarrollo nos encontramos con el desafío de implementar un servidor de estas características podía exceder el alcance del proyecto y no ser inmediatamente compatible con el resto de nuestras tecnologías, por eso, buscamos una alternativa.

Como primer acercamiento nos planteamos el objetivo de elegir un otro sistema de persistencia datos que cumpla los siguientes requisitos.

#### 2.2.5.1 Requisitos de persistencia de datos

- Servicio basado en la nube administrado por un tercero.
- Operaciones de escritura, lectura, actualización son posibles de ejecutar desde un cliente de juego.
- Operaciones de escritura, lectura, actualización son posibles de ejecutar desde un servidor de juego.
- Todas las operaciones tienen un tiempo de respuesta aceptable.
- Administración de permisos y seguridad para interacciones con la base de datos.
- Planes de costo flexibles.

#### 2.2.5.2 Amazon DynamoDB

De las opciones evaluadas el servicio DynamoDB de Amazon [v14] cumple con todos los requisitos y tiene una vía de integración clara con los demás servicios que de Amazon que estamos integrando en nuestro stack de tecnologías.

Una diferencia a notar es que DynamoDB es una base de datos noSQL y Amazon ha implementado el servicio de manera que los costos de uso dependen del volumen de uso. En tal caso las búsquedas en tablas deberían ser evitadas, prefiriendo acceder a datos por indices conocidos lo cual no debería ser un problema.

Digamos que tenemos la siguiente tabla, que lista algunas interacciones recientes de un jugador:

<b>idJugador</b>	<b>fechaUltimoAcceso</b>	<b>idUltimaPartida</b>
1	14 Sep 2022	535
2	17 May 2022	216
...	...	...

Tabla 12. Interacciones recientes de un jugador

Y la siguiente tabla con datos históricos de partidas:

<b>idPartida</b>	<b>idJugador1</b>	<b>idJugador2</b>	<b>idJugadorGanador</b>	<b>fechaPartida</b>	<b>minutosPartida</b>
...	...	...	...	...	...
535	1	5	1	14 Sep 2022	22

...	...	...	...	...	...
-----	-----	-----	-----	-----	-----

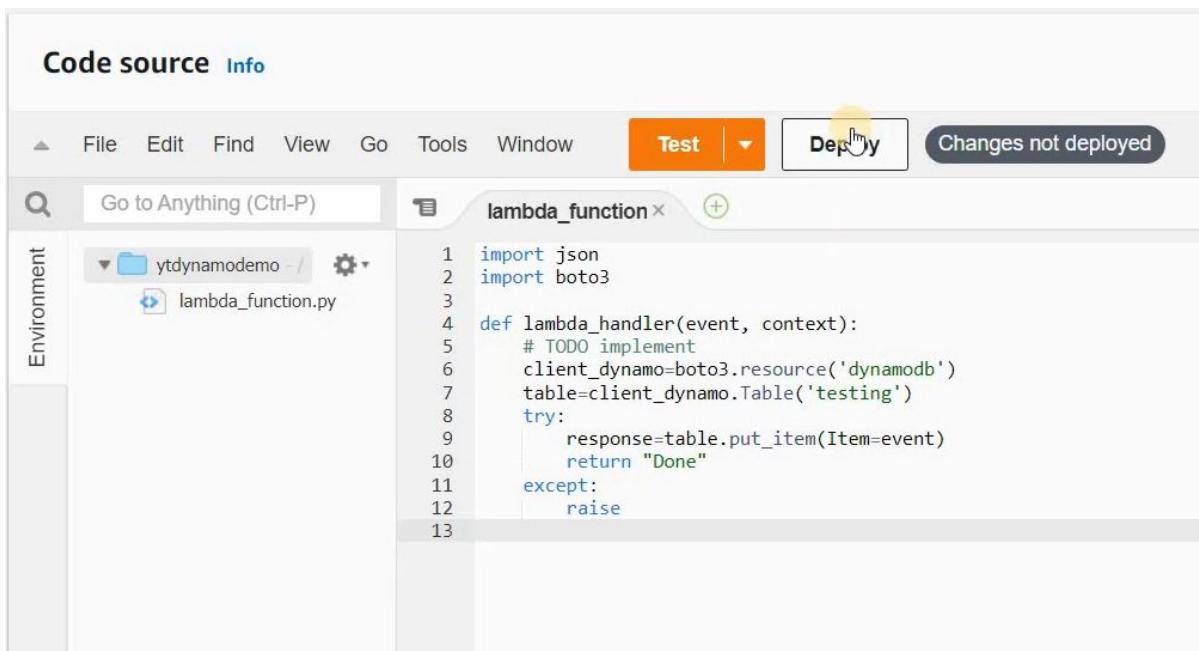
Tabla 13. Histórico de partidas

Un jugador al logearse al juego puede tener como dato su idJugador y de ahí con un par de consultas a índices específicos podemos acceder a información de su última partida sin la necesidad de hacer ninguna búsqueda.

### 2.2.5.3 Integración DynamoDB y Unity

DynamoDB ofrece dos vías inmediatamente válidas para integración con nuestros sistemas de cliente y servidor. Vía funciones Lambda o vía llamadas por código a través de su SDK [v15]. De estas dos opciones nos inclinamos por funciones Lambda ya que nos permiten gestionar permisos de manera más clara y actualizarlas sin tener que recopilar el código del cliente o el servidor.

Las figuras a continuación muestran de un ejemplo práctico [v16] de cómo realizar un insert en una tabla de DynamoDB usando funciones Lambda en Python y las herramientas de testeo *online* que provee Amazon.



```

Code source Info

File Edit Find View Go Tools Window Test Deploy Changes not deployed

Go to Anything (Ctrl-P) lambda_function +
Environment

ytdynamodemo / lambda_function.py

1 import json
2 import boto3
3
4 def lambda_handler(event, context):
5     # TODO implement
6     client_dynamo=boto3.resource('dynamodb')
7     table=client_dynamo.Table('testing')
8     try:
9         response=table.put_item(Item=event)
10        return "Done"
11    except:
12        raise
13

```

Figura 11. Función Lambda que inserta un nuevo ítem en la tabla testing

**Test event**

Invoke your function with a test event. Choose a template that matches the service that triggers your function, or

- New event
- Saved event

Template

```
hello-world
```

Name

```
MyEventName
```

```

1 * {
2     "roll_no": 20,
3     "Name": "Rajdeeop Sil",
4     "University": "VIT",
5     "Address": "Delhi"
6 }
```

Figura 12. Pares de clave valor que forman el ítem a insertar

Items returned (2)							Actions ▾	Create item
	roll_no	Address	Name	University				
<input type="checkbox"/>	10		Soham	KIT				
<input type="checkbox"/>	20	Delhi	Rajdeeop Sil	VIT				

Figura 13. Contenido de la tabla testing luego de insertar el nuevo ítem.

## 2.2.6 Implementación - Game Server

En este segundo sprint la parte más importante de la implementación consistía en llevar a cabo la infraestructura inicial donde se alojaría nuestro juego. Nos propusimos una meta inicial básica, la cual consistía en levantar una instancia de servidor y poder

conectar dos jugadores (clientes) en simultáneo, dónde el jugador que tocará primero la tecla W ganaría la partida. Ésta tarea la realizamos con éxito dentro del tiempo estipulado.

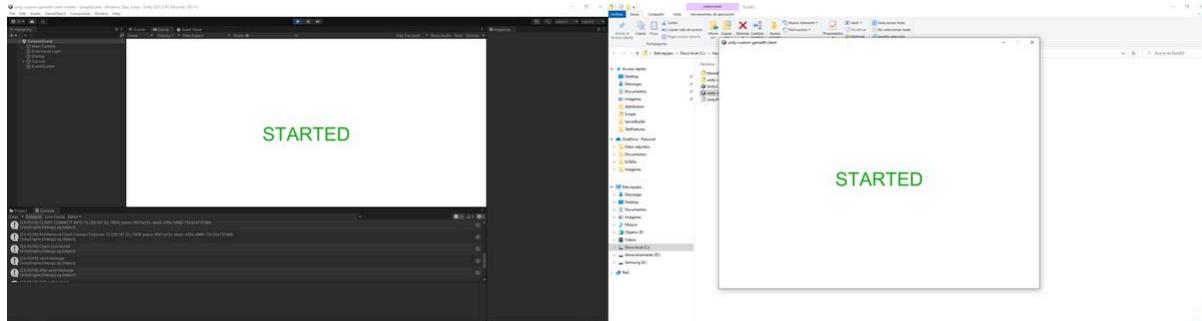


Figura 14. Clientes conectados en simultáneo al servidor AWS Gamelift

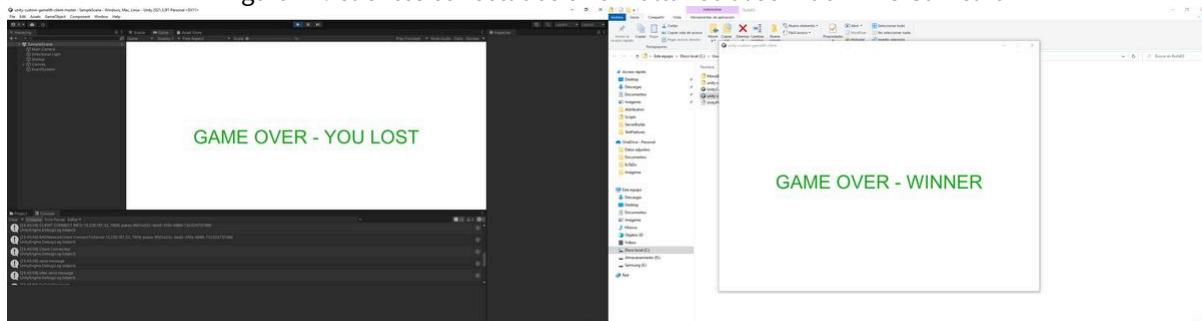


Figura 15. Clientes comunicándose con lógica de juego básica

## 2.2.7 Implementación - Login y Register

Las tareas de este ítem consistían en la implementación del registro e inicio de sesión de los usuarios. Creamos una pantalla donde se permitiera el registro de un usuario, con nombre de usuario, contraseña y mail único que debe ser verificado mediante un link enviado a esa misma dirección, y que permitiera posteriormente a esta verificación, ingresar al sistema con las credenciales proporcionadas. Las tareas secundarias estaban relacionadas con verificar el correcto funcionamiento del repositorio, permitiendo a todos los integrantes trabajar desde sus terminales y poder generar versiones ejecutables de estos primeros prototipos.

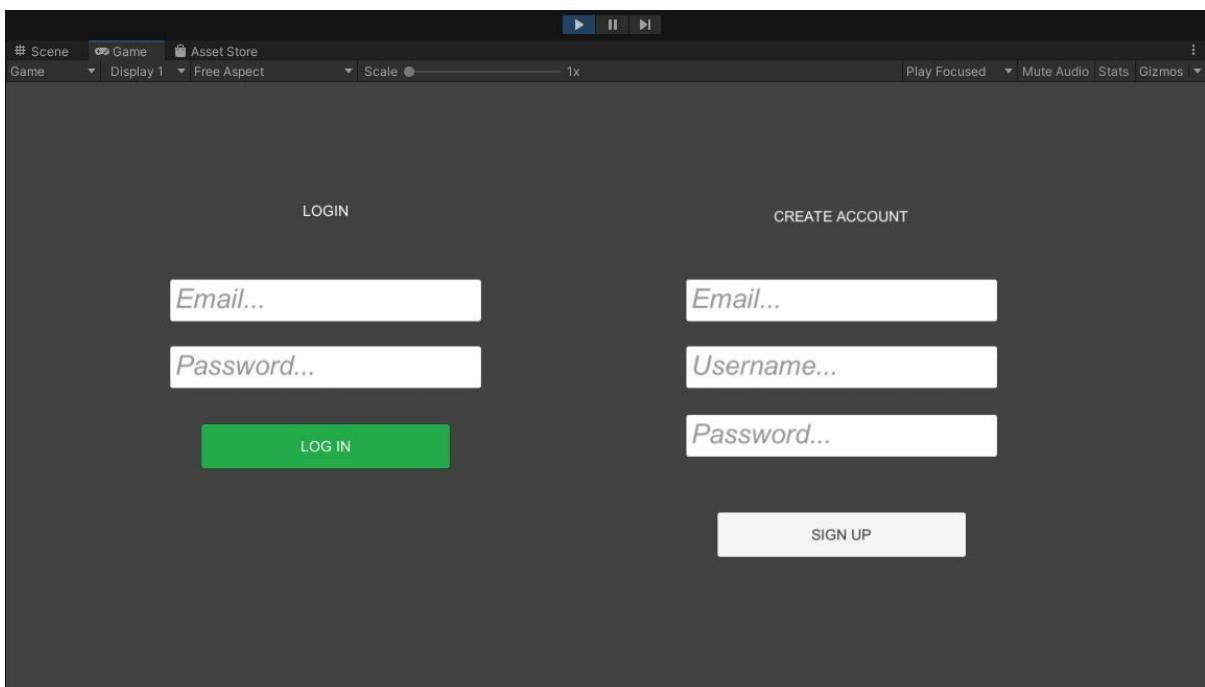


Figura 16. Pantalla inicial de inicio de sesión y registro de usuario

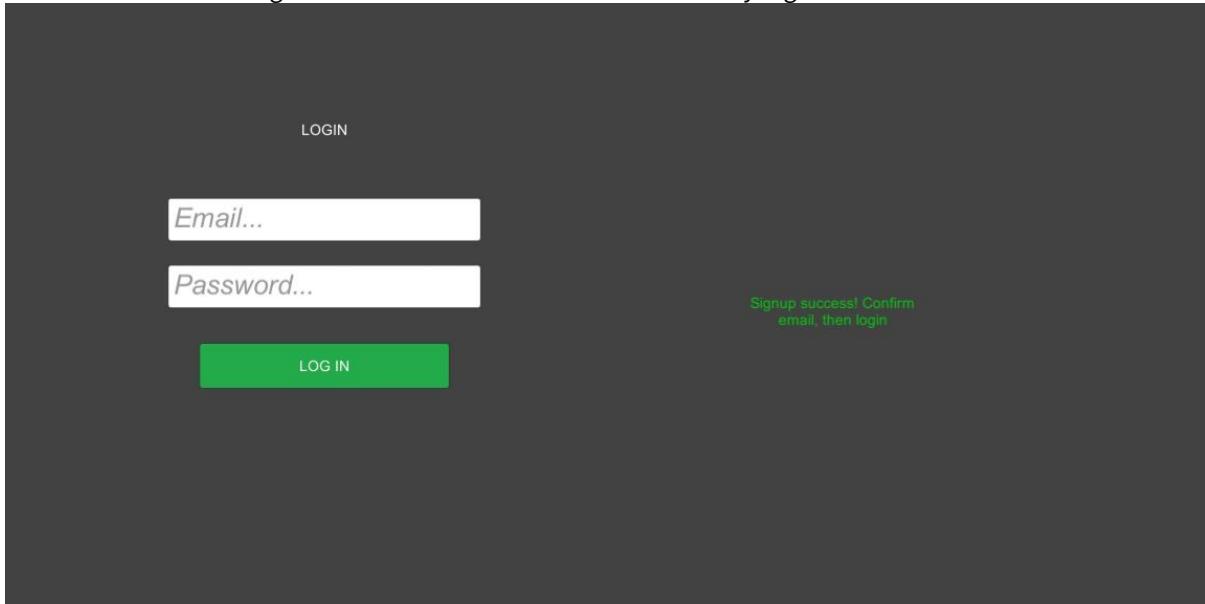


Figura 17. Pantalla de registro de usuario correcto

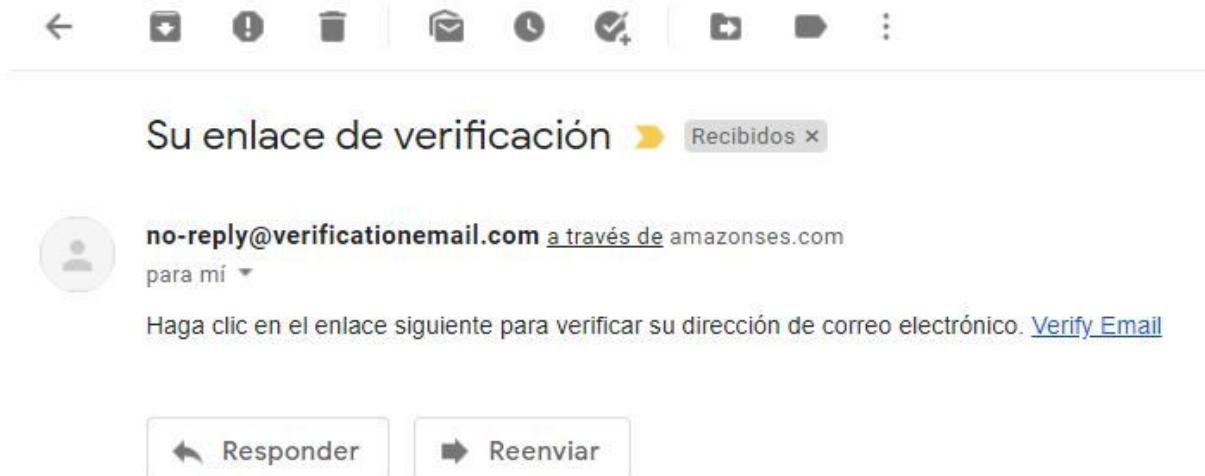


Figura 18. Mail de verificación de correo electrónico

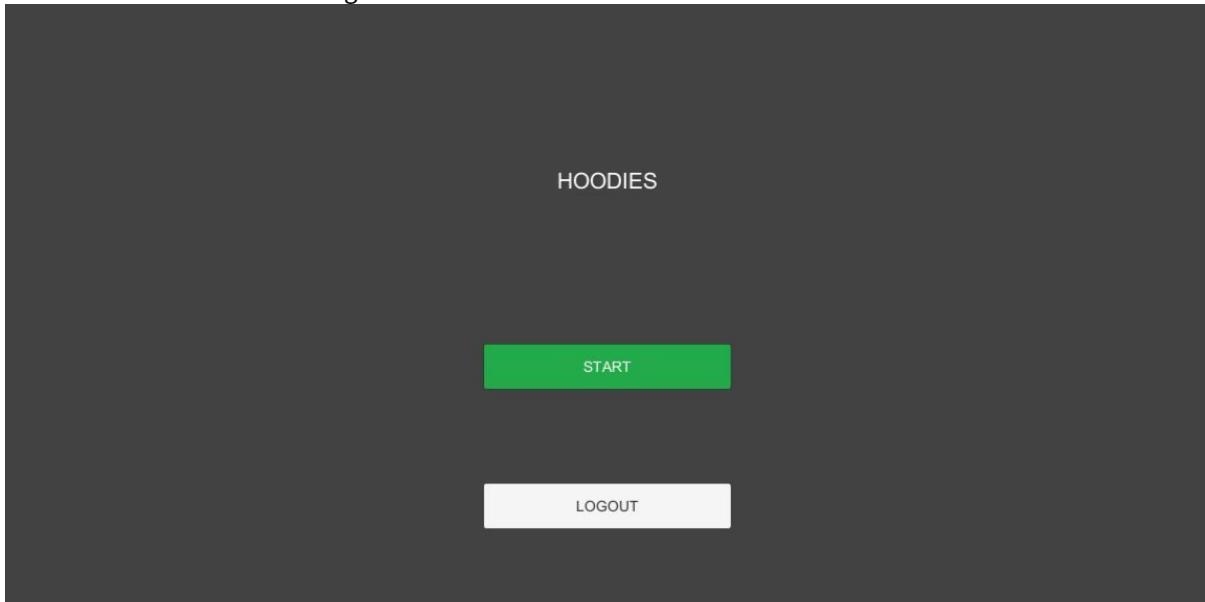


Figura 19. Pantalla de inicio de sesión correcto

## 2.2.8 Stack de Tecnologías

Los distintos cambios que fueron surgiendo en cuanto a tecnologías a utilizar y las variaciones en la implementación de la infraestructura impactan directamente en lo previsto para nuestro [Stack de Tecnologías](#). Teniendo en cuenta nuestro plan actual el diagrama se ve de la siguiente manera.

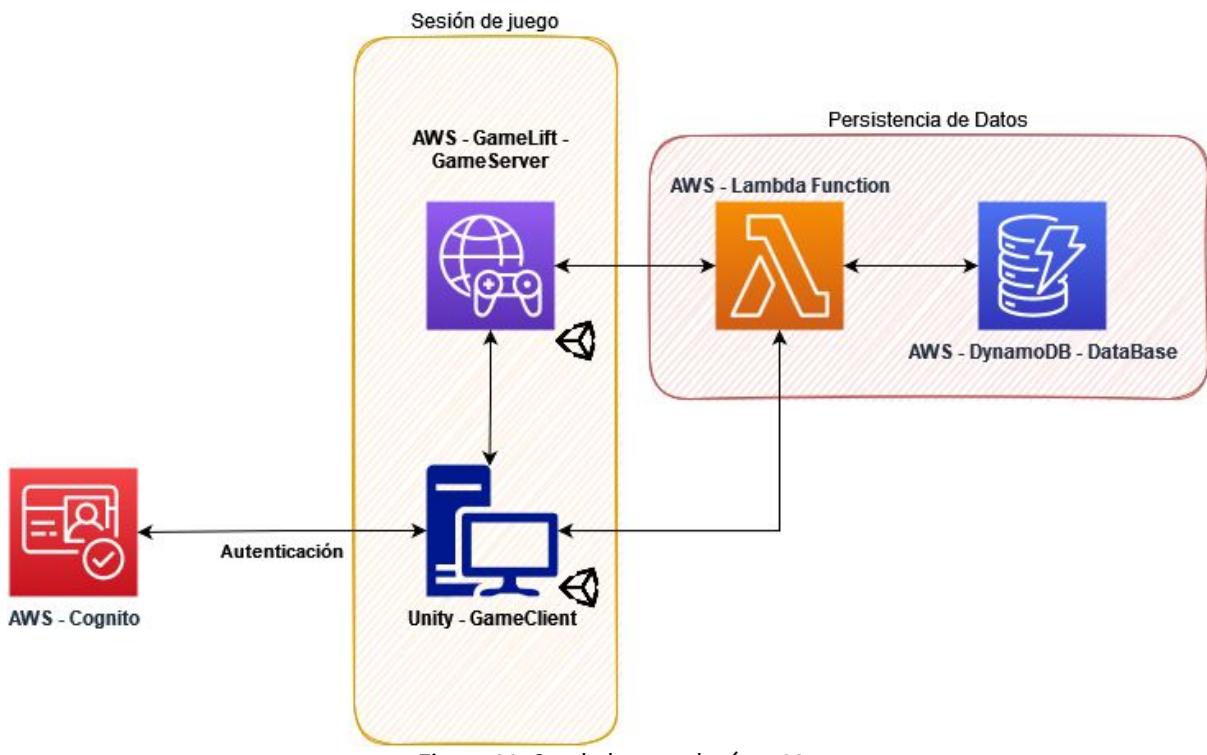


Figura 20. Stack de tecnologías v02

## 2.2.9 Gestión de Proyecto

Como mencionamos inicialmente en el capítulo [Presentación del Grupo Foco](#). Un integrante importante de referencia era *ARF Games Studio* dentro del Subgrupo 1 integrado por profesionales del entorno. Lamentablemente cuando fueron contactados para una reunión de presentación del trabajo realizado hasta ahora nos comunicaron que restricciones de disponibilidad hacían imposible su continuada participación como integrantes del grupo foco.

Esto fue comunicado al tutor que ofreció la buscar un nuevo contacto para cumplir este rol. Forma parte del siguiente sprint terminar de coordinar una comunicación con este nuevo contacto a formar parte del Subgrupo 1.

## 2.3 Sprint 3

### 2.3.1 Resumen

En este *sprint* comenzamos redondeando el tema de deploy del server trabajado en el *sprint 2*, generando una documentación a modo de guía. Luego teníamos planteadas distintas tareas que consistían en investigación y posterior implementación, abarcando varios aspectos tales como la creación de una sala previa de partida (*lobby*), la

implementación del sonido del juego, la implementación de un buen manejo de escenas con información persistente entre las mismas y la investigación de la persistencia de datos con algún servicio en particular de AWS.

### 2.3.2 Tareas

Tareas a llevar a cabo en este *sprint*.

<b>Id</b>	<b>Titulo</b>	<b>Tiempo Estimado/ Real hs</b>	<b>Estado</b>
HOOD-66	Documentar Flujo Unity Servers a Gamelift	3/3	Finalizada
HOOD-67	Implementación Lobby Público y Privado	8/28	Finalizada
HOOD-72	Arquitectura de Sonido	6/7	Finalizada
HOOD-69	Arquitectura de escenas	6/7	Finalizada

Tabla 14. Tareas Sprint 3

#### 2.3.2.1 Reuniones

<b>Objetivo</b>	<b>Fecha</b>	<b>Participantes</b>	<b>Duración</b>
Sprint planning	18 Jun 2022	Agustín, Ramiro y Manuel	3h
Sprint Progress Sync	22 Jun 2022	Agustín, Ramiro y Manuel	15m
Sprint Progress Sync	25 Jun 2022	Agustín, Ramiro y Manuel	1h35m

Tabla 15. Reuniones Sprint 3

### 2.3.3 Implementación - Producción de Documentación Interna

En los *sprints* anteriores realizamos muchas tareas respectivas a la infraestructura, como lo es el *deploy* del servidor que alojará las sesiones de juego. Particularmente, la tarea de levantar un servidor requiere una serie de pasos precisos y difíciles de recordar, es por esto que consideramos pertinente documentar una guía en nuestra *wiki* de

Confluence. De esta forma todos los integrantes tiene acceso a la misma y pueden recurrir a ella cuando necesiten levantar un servidor de *test* o producción.

#### 2.3.4 Implementación - Lobby Público y Privado

Para este *sprint* teníamos planteada al tarea de investigación e implementación de un *lobby* para las partidas tanto públicas como privadas. Investigamos las distintas alternativas que teníamos. Lamentablemente no hay mucha información ni documentación al respecto, lo cual llevó a que está tarea se extendiera más de lo previsto. De la investigación surgieron dos alternativas para implementar la búsqueda de partidas y creación de *lobby*, una de ellas es utilizando un servicio de AWS llamado “Flex Match”. Este es un servicio de emparejamiento para juegos multijugador, que permite definir un conjunto de reglas personalizado. Si bien consideramos que este servicio nos permite cumplir con nuestros requerimientos, la complejidad y el exceso de funciones de las cuales no haremos uso hicieron que desistamos de esta opción. La siguiente alternativa es utilizar la misma API de GameLift para crear y buscar las sesiones de juego como veníamos haciendo hasta el momento. Encontramos que mediante algunas configuraciones y parámetros, podríamos implementar lo justo y necesario para nuestro *lobby*. Si bien sigue siendo algo complejo, lo es en un menor nivel y cumple exactamente con nuestros requerimientos.

Crear un *lobby* para nosotros implica crear una partida en un estado de sala de espera o preparación, dónde los jugadores puedan prepararse para jugar. Luego de investigar las alternativas, optamos por utilizar la API de GameLift. Para ejecutar las distintas operaciones implementamos una primera interfaz con las secciones “crear”, “unirse” y buscar:

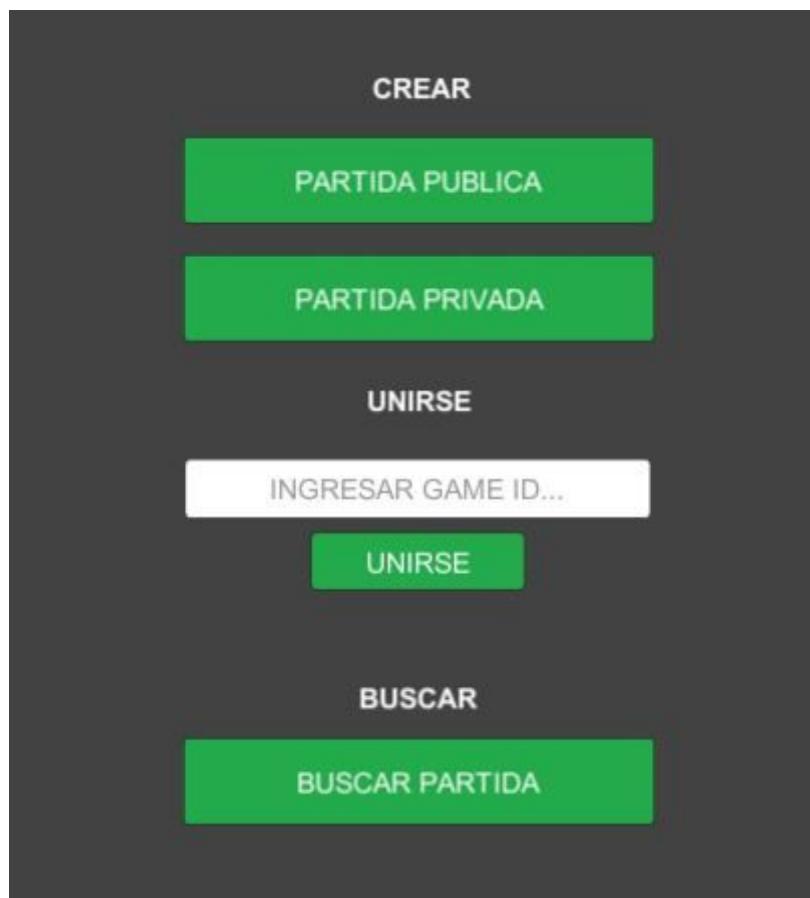


Figura 21. Interfaz temporal pre-lobby

Tanto la partida pública como privada implican la creación de una `GameSession` mediante una *request* a la API. Ésta consiste en una instancia del juego que se ejecuta en el servidor anteriormente levantado en Amazon GameLift, los jugadores pueden crear o unirse a una sesión de juego mediante la creación de una `PlayerSession`. Tanto el objeto `GameSession` como el `PlayerSession` tienen en sus propiedades información de la partida y del jugador respectivamente. Cabe destacar que esta información se genera para cada partida, por ejemplo, un mismo jugador/cliente va a tener un `playerId` distinto en partidas distintas. Hasta el momento no habíamos profundizado en estos términos ya que simplemente los veníamos utilizando para unir a los dos jugadores que primero ingresaban.

Las llamadas principales que utilizamos son las siguientes:

#### 2.3.4.1 CreateGameSession

```
{  
  "AliasId": "string",  
  "CreatorId": "string",  
  "FleetId": "string",  
  "GameProperties": [  
    {  
      "Key": "string",  
      "Value": "string"  
    }  
  ],  
  "GameSessionData": "string",  
  "GameSessionId": "string",  
  "IdempotencyToken": "string",  
  "Location": "string",  
  "MaximumPlayerSessionCount": number,  
  "Name": "string"  
}
```

#### 2.3.4.2 CreatePlayerSession

```
{  
  "GameSessionId": "string",  
  "PlayerData": "string",  
  "PlayerId": "string"  
}
```

#### 2.3.4.3 SearchGameSession

```
{  
  "AliasId": "string",  
  "FilterExpression": "string",  
  "FleetId": "string",  
  "Limit": number,  
  "Location": "string",  
  "NextToken": "string",  
  "SortExpression": "string"  
}
```

Al momento de crear las partidas, para diferenciar las públicas de las privadas, ingresamos en la *request* una `GameProperty` con `Key = "type"` y `Value = "public"` o `"private"` respectivamente. Esto nos permite diferenciarlas, y al momento de buscar una partida poder buscar solamente las partidas públicas. Esto se realiza mediante la propiedad `FilterExpression` en la *request* de `SearchGameSession`, donde podemos filtrar por la propiedad de juego mencionada anteriormente.

```
searchGameSessionRequest.FilterExpression =
“gameSessionProperties.type='public'
AND hasAvailablePlayerSessions=true”;
```

Con el fin de desarrollar de manera más ágil estas operaciones y comportamiento, creamos una interfaz temporal, que si bien no se ajusta del todo a los *mockups* definidos en un principio, cumple con las necesidades básicas para poder probar las funciones desarrolladas.

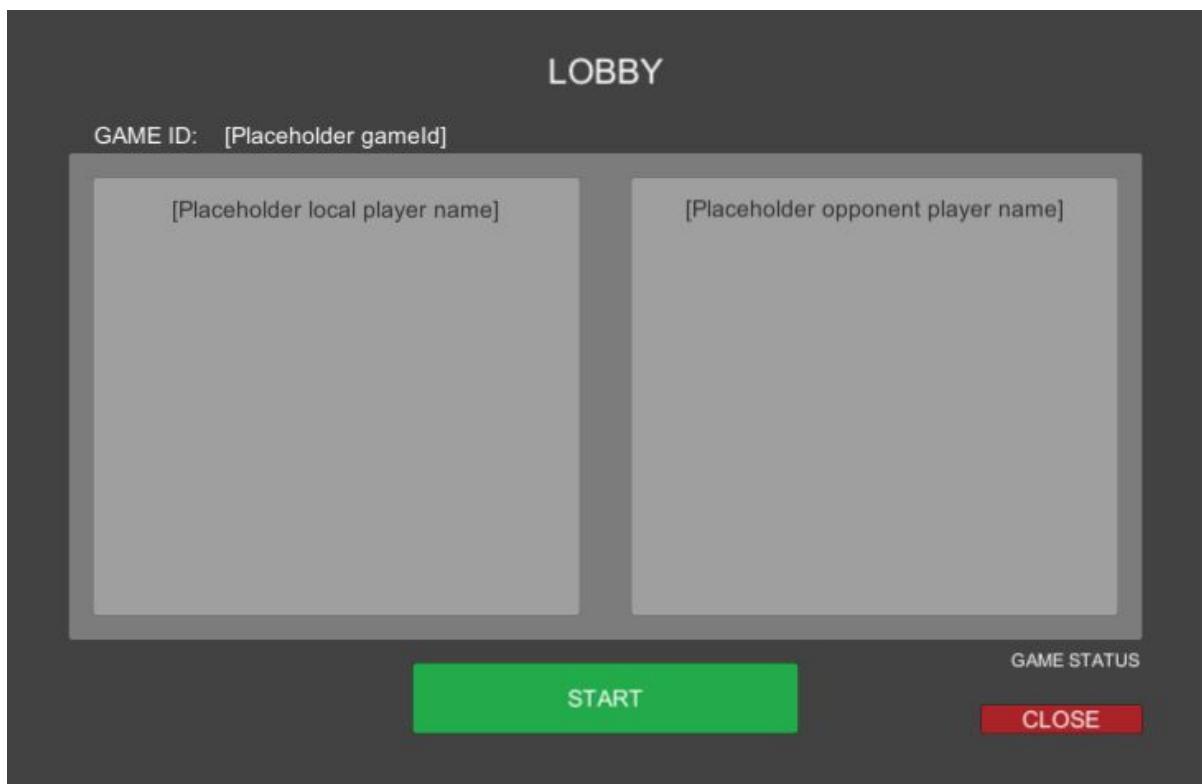


Figura 22. Interfaz de testing para funcionalidades de lobby

### 2.3.5 Implementación - Arquitectura de Sonido

Los videojuegos son una experiencia audiovisual, por lo que tener una arquitectura robusta para los sonidos es de gran prioridad para el proyecto.

Para comenzar se creó un objeto Sound que contiene el clip de audio es sí y reproduce este audio al activarse el objeto en la escena y lo deja de reproducir cuando se desactiva. Por otro lado, para contener todos los distintos audios que existirán en el juego, se decidió crear un [ScriptableObject](#) [v17]

) que contiene la información necesaria de cada audio.

Dado a que un sonido que ocurre luego de una acción que ocurre en el juego y el sonido que se reproduce con una escena (música y sonido ambiental) se comportan de manera distinta, también fue necesario crear un [ScriptableObject](#) que contenga la información de los audios dedicados a este último rol mencionado.

Todo lo mencionado anteriormente permite el manejo y la organización de los datos asociados al audio del juego, pero no permite el control de la reproducción del mismo.

Para este fin se construyó un [AudioManager](#). El [AudioManager](#) contiene dos diccionarios que se pueblan respectivamente con los datos de los dos [ScriptableObjects](#) ya mencionados y contiene la lógica de cómo se reproducirán los sonidos.

A su vez, para poder configurar muchos aspectos de la reproducción del sonido como su volumen, Unity requiere la creación de mezcladores de audio. Estos mezcladores permiten reproducir sólo un sonido por mezclador, algo que no es problemático para la música y el sonido ambiental pero sí para los efectos de sonido — no es difícil imaginar una situación donde se necesite reproducir el sonido de un ataque y el de una unidad recibiendo ese ataque, por ejemplo.

Este problema llevó a la siguiente solución, ya que los usuarios deberían poder silenciar los efectos de sonido si así lo desearan.

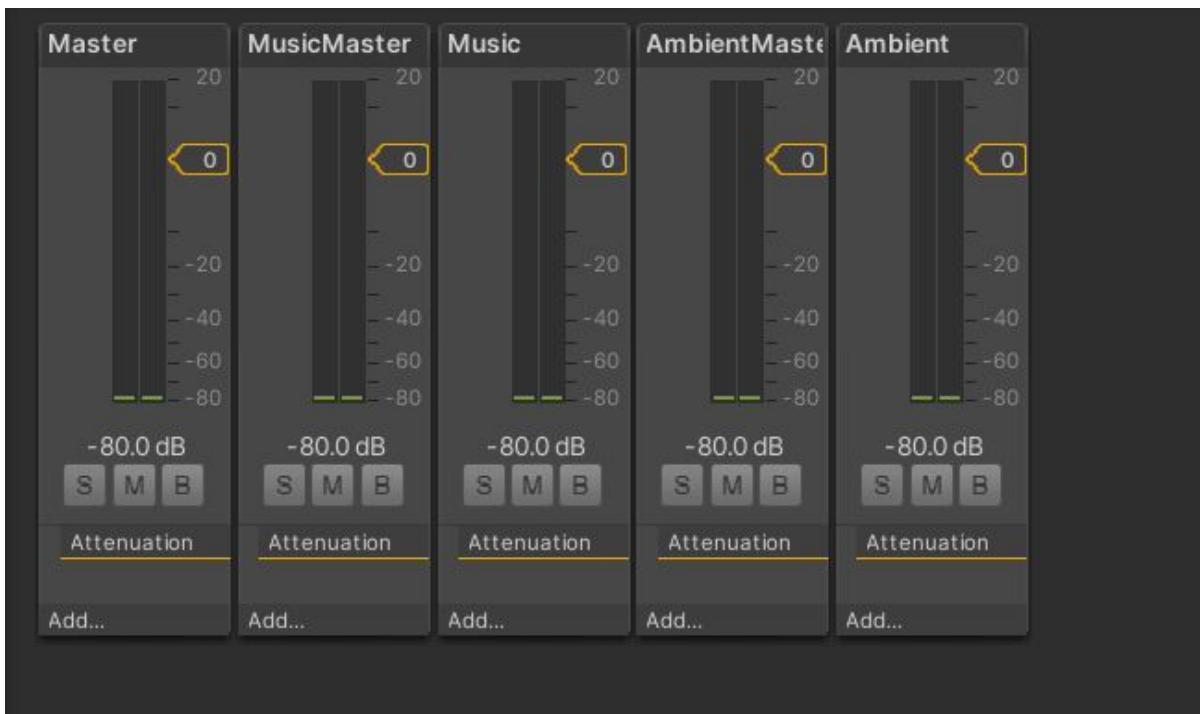


Figura 23. Mezcladores de audio

```

public void PlaySound(SoundName soundName)
{
    if (_soundDictionary.TryGetValue(soundName, out SoundItem soundItem) &&
mySoundPrefab != null)
    {
        GameObject soundGameObject =
myPoolManager.ReuseObject(mySoundPrefab, Vector3.zero,
Quaternion.identity); // We retrieve the generic 'Sound' gameobject from the
pool

        Sound sound = soundGameObject.GetComponent<Sound>();

        sound.SetSound(soundItem, mySFXVolume);

        soundGameObject.SetActive(true); // It automatically plays the
sound because it plays on Awake
        StartCoroutine(DisableSound(soundGameObject,
soundItem.soundClip.length)); // Disable the sound after it's done playing
    }
}

public void ChangeSFXVolume(float volume)
{
    mySFXVolume = volume;
}

```

Código que maneja la reproducción de efectos de sonido (`AudioManager`)

Como se puede apreciar, no existe un mezclador de audio de efectos de sonido, pero sí existe una propiedad `mySFXVolume` en `AudioManager`. Esta propiedad es usada antes de la reproducción del sonido para ajustar su volumen. De la perspectiva del usuario, esto funciona exactamente igual que ajustar el volumen de un canal dedicado en el mezclador de audio.

### 2.3.6 Implementación - Arquitectura de escenas

La escena es el objeto base donde se ejecuta el juego, contiene otros objetos como cámaras, jugadores, elementos interactivos. Como mínimo siempre debe existir una escena en ejecución en cada proyecto de Unity, pero pueden haber más. Salvo que se especifique en el código de un objeto, todos los objetos de una escena se crean al activarla y se destruyen al desactivarla.

Dado que es necesario conservar información entre las distintas escenas que existen en el juego, necesitamos crear un sistema para administrar lo que se debe destruir y lo que no.

La solución más óptima y escalable que encontramos fue crear una escena persistente, que nunca será desactivada. Esta escena contiene todos los datos que serán compartidos y referenciados en el resto de las escenas.

Para permitir que se cambien de escenas también se debió crear una clase `SceneManager`, que se encarga de cargar y descargar escenas dependiendo de la situación. Las escenas se cargan aditivamente, lo que quiere decir que su información se añade a la escena persistente, permitiendo un flujo de datos eficiente.

Para darle una mayor flexibilidad al código, se decidió crear una clase `EventHandler`, que usa la funcionalidad nativa de C# de *Events* [v18]. Esto permite que las funciones suscriptas al evento se llamen cuando el evento sucede.

El uso de eventos tiene innumerables usos que son aplicados en otras partes del proyecto, pero en el caso de las escenas, cuando se cambia a una escena por otra se llama al evento `CallAfterSceneLoadEvent()`. Si necesitamos que algo suceda apenas se cambia de escena, podemos suscribirnos a este evento.

## Ejemplo de transición de escenas

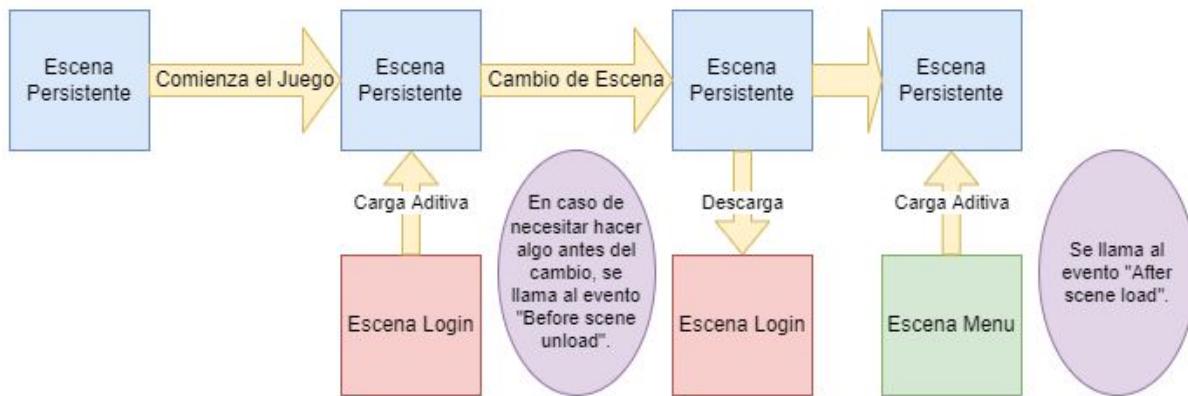


Figura 24. Ejemplo de transición de escenas

## 2.4 Sprint 4

### 2.4.1 Resumen

Para este *sprint* teníamos previsto llevar a cabo la implementación de algunos de los *mockups* de escenas, tales como la sección del menu principal, la sección de los mazos y la sección de configuración (opciones). A su vez mejorar las interfaces temporales que habían sido creadas con motivo de prueba de concepto o *testing*. Por último continuamos con tareas de investigación de otros servicios de AWS necesarios.

### 2.4.2 Tareas

Tareas a llevar a cabo en este *sprint*.

<b>Id</b>	<b>Título</b>	<b>Tiempo Estimado/ Real hs</b>	<b>Estado</b>
HOOD-73	Estructura de UI	6/4	Finalizada
HOOD-74	Login Scene	4/4	Finalizada
HOOD-75	Menu Scene	8/6	Finalizada

HOOD-76	Integración AWS Lambda y .NET	4/5	Finalizada
---------	-------------------------------	-----	------------

Tabla 16. Tareas Sprint 4

#### 2.4.2.1 Reuniones

Objetivo	Fecha	Participantes	Duración
Sprint planning	03 Jul 2022	Agustín, Ramiro y Manuel	2h15m
Sprint sync, pair programming session	13 Jul 2022	Agustín, Ramiro y Manuel	2h
Pair programming session	14 Jul 2022	Ramiro y Manuel	2h
Pair programming session	15 Jul 2022	Ramiro y Manuel	2h30m

Tabla 17. Reuniones Sprint 4

#### 2.4.3 Implementación - Estructura de UI

En Unity, toda la UI debe contenerse en *canvas*, son los Objetos base de la UI. Los *canvas* están optimizados para renderizar objetos 2D y manejar input de usuario. Una escena puede tener múltiples *canvases*. En un *canvas* se pueden ubicar objetos como textos, símbolos y botones.

En cuanto a las distintas escenas, para el manejo de los varios estados de cada una llegamos al siguiente esquema, aplicando el patrón experto.

- Cada escena contiene un script del tipo *SceneUIManager* ('Scene' siendo el nombre de la escena en cuestión)
- Cada *canvas* de cada escena contiene un script del tipo *CanvasUIManager* ('Canvas' siendo el nombre del *canvas* en cuestión)

##### 2.4.3.1 SceneUIManager

Este *script* se encarga de controlar que *canvas* está activo y de iniciar la transición a otras escenas.

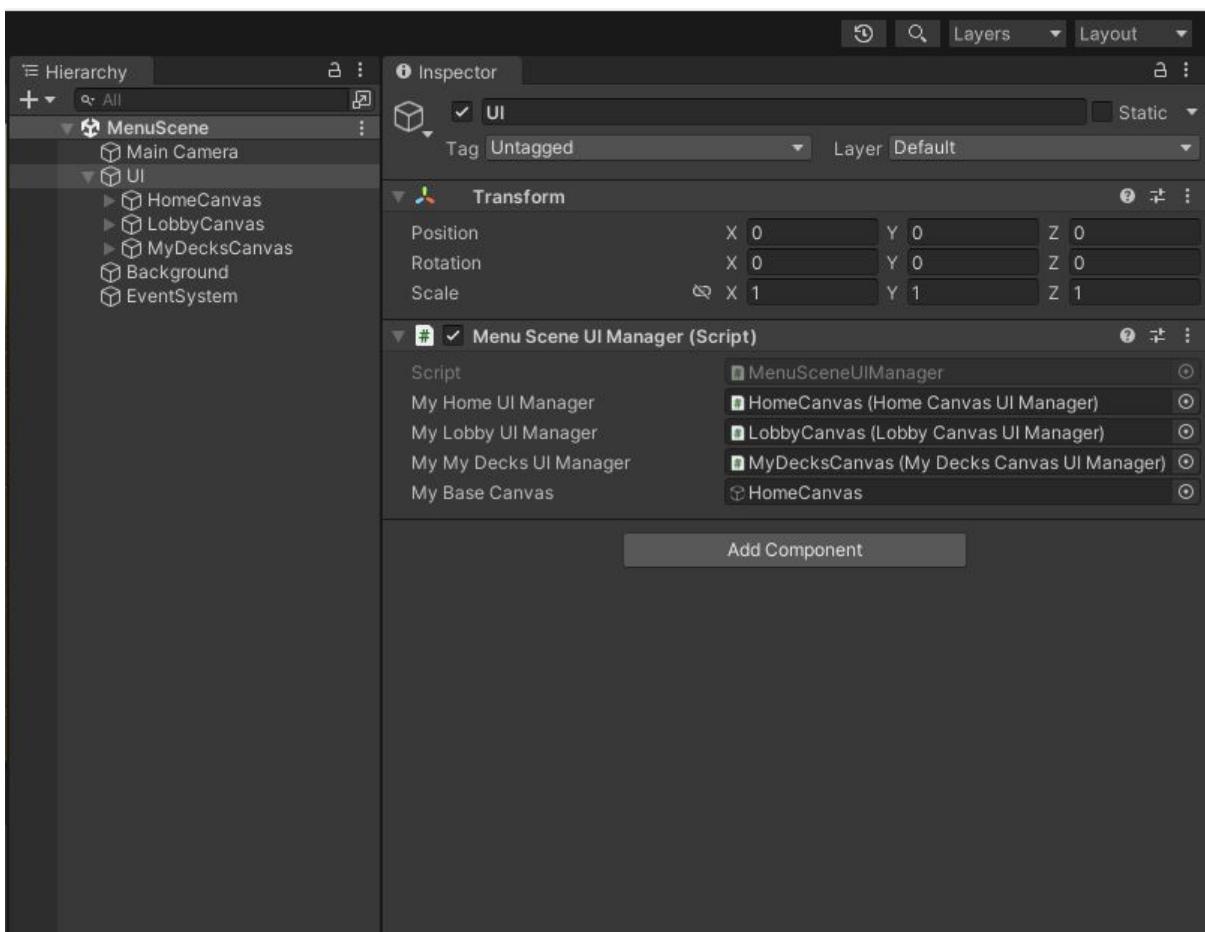


Figura 25. Ventanas de Unity demostrando un SceneUIManager

#### 2.4.3.2 CanvasUIManager

Este *script* se encarga de controlar que objetos del canvas mostrar, ocultar u actualizar textos y de manejar la interacción de los usuarios con botones o diálogos de entrada.

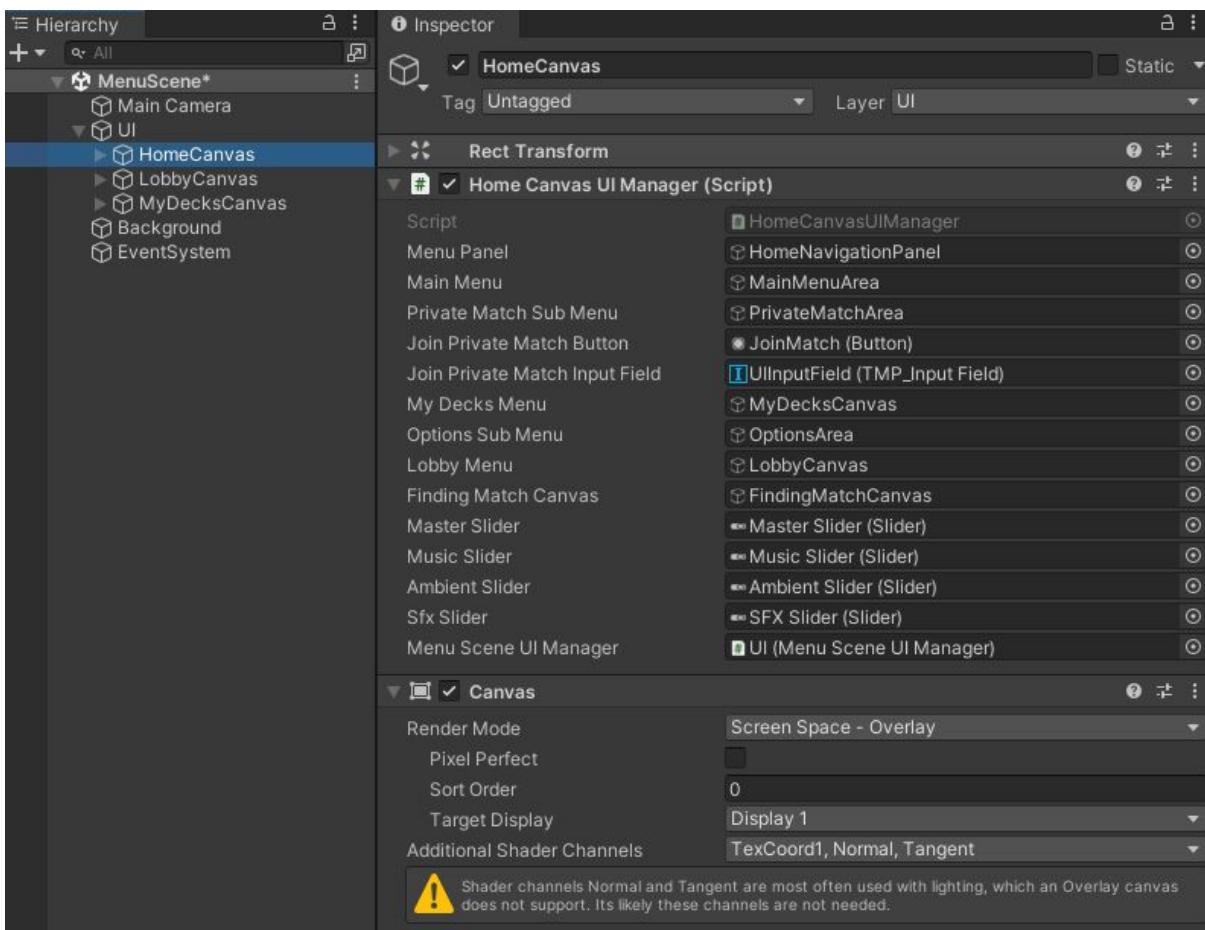


Figura 26. Ventanas de Unity Mostrando un CanvasUIManager

#### 2.4.4 Implementación - Login Scene

Hasta este *sprint*, se usaba una UI de *login* temporalia que no aprovechaba las funcionalidades de Unity ni respetaba la temática del juego. Por ejemplo, usando la funcionalidad de *prefabs* [g9], es posible reutilizar botones y campos de texto en todo el proyecto. Algo que no sólo ahorra tiempo, sino que también contribuyen a la uniformidad temática de los menús.

# HOODIES

LOGIN      SIGN UP

*Email...*

*Password...*

**LOG IN**

*Email...*

*Username...*

*Password...*

**SIGN UP**

This figure shows a composite screenshot of a mobile application's login and sign-up screen. The top half displays the 'LOGIN' section with an email input field containing 'Email...', a password input field containing 'Password...', and a black 'LOG IN' button. The bottom half displays the 'SIGN UP' section with an email input field containing 'Email...', a placeholder 'Username...' above a password input field containing 'Password...', and a black 'SIGN UP' button. All text is in a light blue color.

Figura 27. S01\_Login y S02\_Register fueron combinados en una sola pantalla.

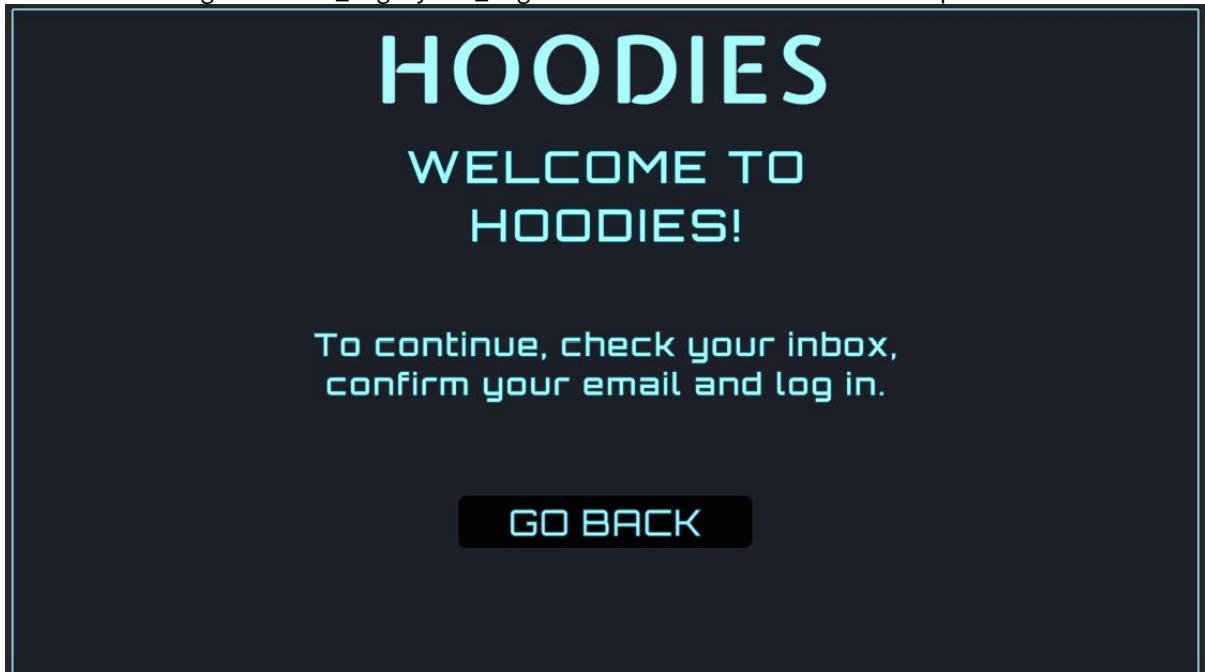


Figura 28. Una vez que el usuario elige Sign Up con los datos adecuados, recibe esta pantalla.

## 2.4.5 Implementación - Menu Scene

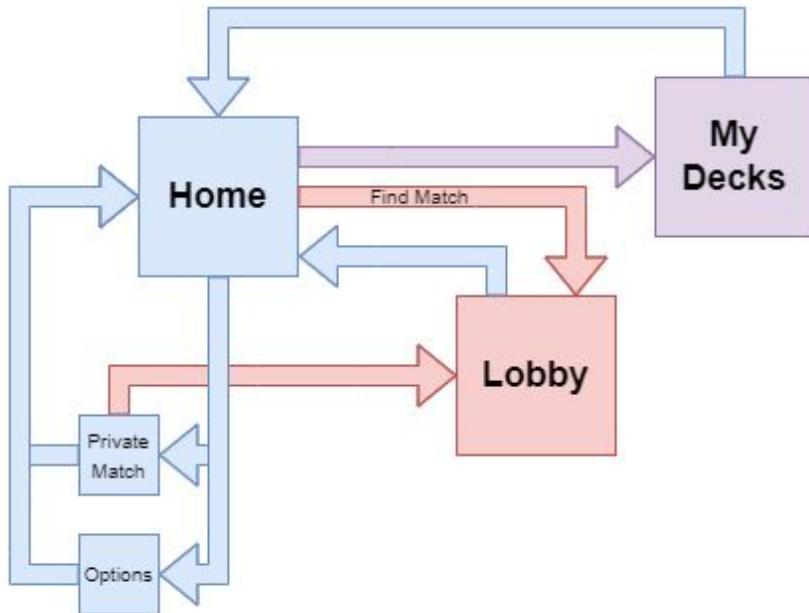


Figura 29. Navegación Interna de MenuScene

Basándonos en los *mockups* creados en el anteproyecto, se creó un menú principal donde es posible navegar entre sus distintas pantallas. En este momento muchas de las funcionalidades más complejas, como encontrar una partida, o armar un mazo, no estaban disponibles pero el menú fue creado de una manera para que implementar estas funcionalidades sea lo más sencillo posible.

La UI de esta escena se dividió en tres partes: `Home` , `Lobby` y `MyDecks` .

### 2.4.5.1 Home

Este menú es el primero con el que un jugador se encuentra después de loguearse. Dentro de este menú existen sub-menús que permiten al usuario acceder a distintas opciones. Para acceder tanto a estos sub-menús, como a las otras partes mencionadas, se utilizan botones. A diferencia de lo que ve en los *mockups* incluidos en el anteproyecto, se optó por eliminar el sub-menú *S05\_PlayMenu* al ser considerado superfluo.

# HOODIES

Find Match

Private Match

My Decks

Options

Logout

Quit Game

Figura 30. Home - Main Menu

# PRIVATE MATCH

Host Match

Join Match

*Enter match ID...*

Go Back

Figura 31. Home - Private Match



Figura 32. Home - Options

### 2.4.5.2 Lobby

Este menú es el que el jugador encuentra una vez que pudo crear o unirse a una partida. Aquí se muestra y se esconde información dependiendo de qué tipo de partida es y lo que sucede en el `lobby`. Por ejemplo, en una partida pública, no existirá y por tanto no se mostrará el *private match ID*.

Utilizando el botón de *Leave* el usuario podrá cancelar la partida y volverá a `Home`, salvo que el tiempo mostrado en el centro de la pantalla llegue a 0, algo que imposibilitará la cancelación de la partida.



Figura 33. Lobby

### 2.4.5.3 My Decks

Este menú contiene todas las cartas disponibles para cada usuario y permite la creación, edición y borrado de mazos. Al presionar un botón con el nombre del *deck* ya creado, un usuario puede consultar ese mazo. Presionando el botón de *Edit* o *Save*, se podrá editar y guardar respectivamente. Con el botón *New Deck...* el usuario creará un mazo en limpio que luego podrá editar y guardar.



Figura 34. My Decks

#### 2.4.6 Investigación - Integración AWS Lambda y .NET

Lambda es un servicio de Amazon que nos permite la ejecución de funciones atómicas definidas en contenedores discretos *hosteados* en la nube. Estas funciones permiten enviar mensajes a otros servicios o realizar procesamiento de baja intensidad sin necesidad de *hostear* un servidor dedicado con un programa ejecutando todo el tiempo que contenga esta función.

Al solicitar acceso a una Lambda, AWS activa un contenedor virtual que aloja nuestra función, ejecuta la función, y luego espera unos segundos antes de volver desactivarse y liberar el recurso.

##### 2.4.6.1 Cloud9

Para crear y editar estas funciones, AWS ofrece un entorno de desarrollo integrado en el *web browser* llamado Cloud9, pero este IDE solo admite trabajar lenguajes interpretados como Node.js and Python.

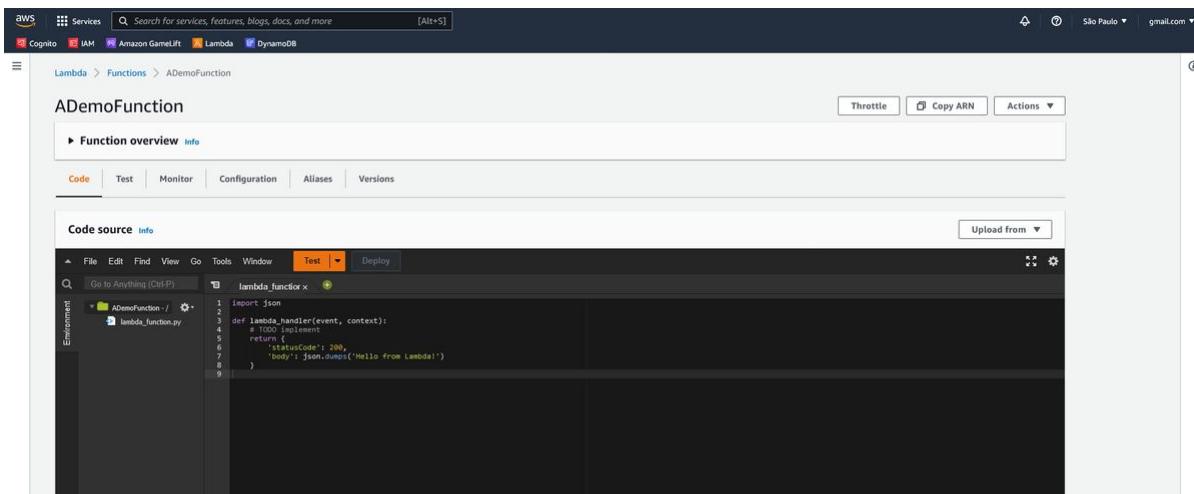


Figura 35. Cloud9 IDE

#### 2.4.6.2 AWS Toolkit

Afortunadamente existe una integración con Visual Studio 2022 para .NET que permite tener acceso a un IDE más completo, testeo local, e integración con bibliotecas de otros servicios de Amazon con los que quisiéramos comunicarnos con nuestra Lambda.

Agregando la extensión gratuita de Visual Studio “AWS Toolkit for Visual Studio 2022” [v19] podemos crear proyectos de lambda fácilmente.

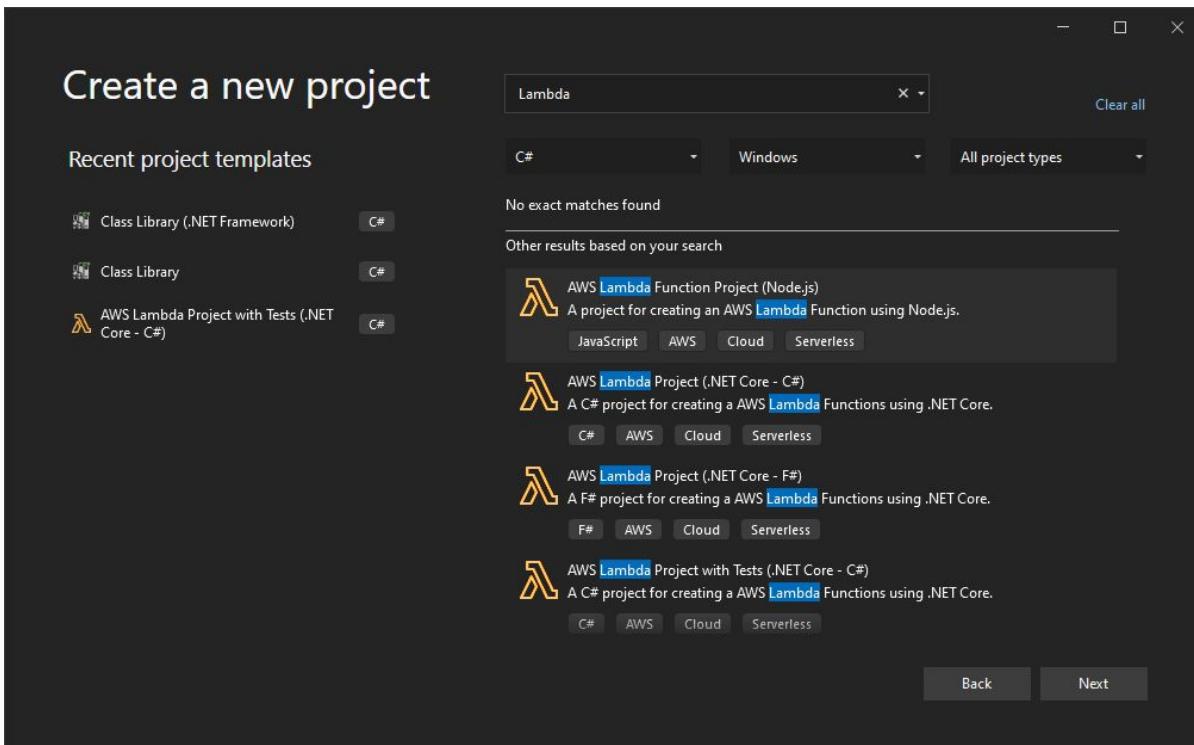


Figura 36. Opciones de nuevos proyectos Lambda en VS Community 2022

### 2.4.6.3 Partes principales de una Lambda

La plantilla por defecto incluye dos archivos de importancia.

#### Descripción

Un archivo que describe nuestra función llamado `aws-lambda-tools-defaults.json` como el siguiente.

```
{  
    "Information": [  
        "This file provides default values for the deployment wizard inside Visual  
        Studio and the AWS Lambda commands added to the .NET Core CLI.",  
        "To learn more about the Lambda commands with the .NET Core CLI execute the  
        following command at the command line in the project root directory.",  
        "dotnet lambda help",  
        "All the command line options for the Lambda command can be specified in  
        this file."  
    ],  
    "profile": "default",  
    "region": "sa-east-1",  
    "configuration": "Release",  
    "function-runtime": "dotnet6",  
    "function-memory-size": 256,  
    "function-timeout": 30,  
    "function-handler":  
        "DotNETDemoFunction::DotNETDemoFunction.Function::FunctionHandler"  
}
```

Componentes de importancia que integran la descripción.

profile

Nombre de usuario de nuestra cuenta de AWS

region

Región donde planeamos alojar nuestra Lambda

function-memory-size

Cantidad de memoria que planeamos reservar para la ejecución de nuestra Lambda

function-timeout

Tiempo en segundos antes de la desactivación de la función una vez ejecutada

function-handler

Declaración de el *namespace* y punto de entrada para la ejecución de nuestra función.

Definición

Y un archivo que contiene la definición de nuestra función llamado `Function.cs` como el siguiente:

```
using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be
// converted into a .NET class.
[assembly:
LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace DotNETDemoFunction;

public class Function
{

    /// <summary>
    /// A simple function that takes a string and does a ToUpper
    /// </summary>
    /// <param name="input"></param>
    /// <param name="context"></param>
    /// <returns></returns>
    public string FunctionHandler(string input, ILambdaContext context)
    {
        // Define your function behaviour here
        return input.ToUpper();
    }
}
```

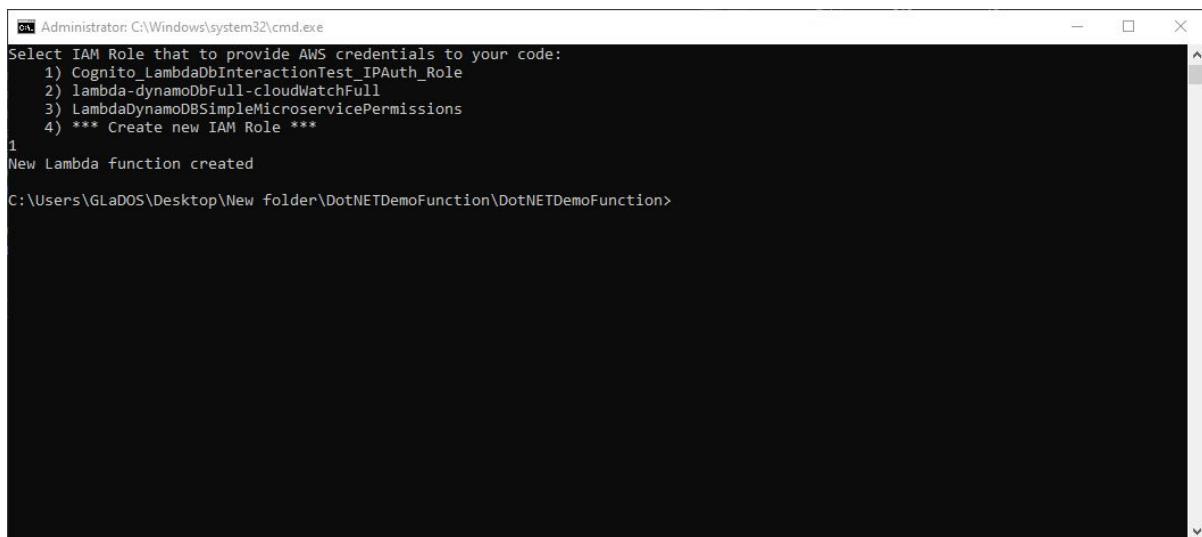
Podemos editar el nombre de las funciones pero debemos reflejar los cambios en el archivo `aws-lambda-tools-defaults.json`

#### 2.4.6.4 Deploying Lambda

Una vez creada nuestra función se puede subir a nuestra cuenta de AWS usando el siguiente comando de consola con AWS CLI `dotnet lambda deploy-function DotNETDemoFunction`

Para que el comando funcione la consola tiene que estar ejecutando en el directorio donde está definida la función y usar su nombre.

A través de la consola podemos elegir un rol disponible para asociarle o crear uno nuevo, este rol definirá que usuarios puede acceder a ejecutar la función y sus permisos.

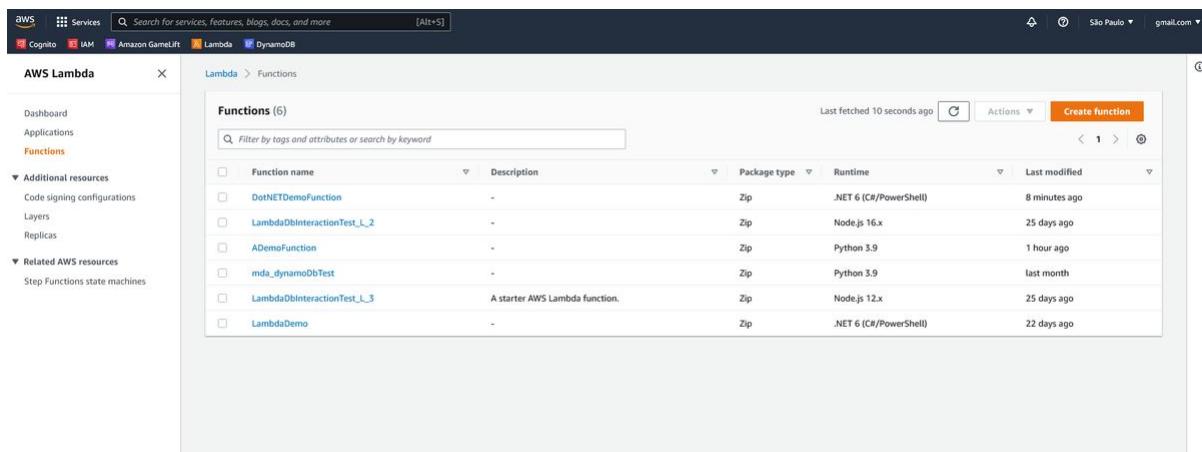


```
Administrator: C:\Windows\system32\cmd.exe
Select IAM Role that to provide AWS credentials to your code:
 1) Cognito_LambdaDbInteractionTest_IAuth_Role
 2) lambda-dynamoDbFull-cloudWatchFull
 3) LambdaDynamoDBSimpleMicroservicePermissions
 4) *** Create new IAM Role ***
1
New Lambda function created

C:\Users\GLaDOS\Desktop>New folder\DotNETDemoFunction\DotNETDemoFunction>
```

Figura 37. Consola CLI luego de la creación exitosa de una función Lambda

Podemos ver a través de la interfaz web que nuestra nueva función ya se encuentra disponible.



The screenshot shows the AWS Lambda service in the AWS Management Console. The left sidebar has 'AWS Lambda' selected under 'Services'. The main area is titled 'Functions' and shows a list of six functions:

Function name	Description	Package type	Runtime	Last modified
DotNETDemoFunction	-	Zip	.NET 6 (.NET/PowerShell)	8 minutes ago
LambdaDbInteractionTest_1_2	-	Zip	Node.js 16.x	25 days ago
ADemoFunction	-	Zip	Python 3.9	1 hour ago
mda_dynamoDbTest	-	Zip	Python 3.9	last month
LambdaDbInteractionTest_1_3	A starter AWS Lambda function.	Zip	Node.js 12.x	25 days ago
LambdaDemo	-	Zip	.NET 6 (.NET/PowerShell)	22 days ago

Figura 38. Interfaz web de AWS Lambda listando las funciones disponibles.

#### 2.4.6.5 Integración de Lambda con Unity

Una vez definida nuestra función Lambda, usuarios que cumplan el rol apropiado pueden hacer llamadas a la función desde dentro de el proyecto de Unity usando el siguiente patrón.

```
public class LambdaManager : MonoBehaviour
{
    private CognitoAWSCredentials myCredentials;
    private RegionEndpoint myRegionId;
    private string myLambdaFunctionName = "DotNETDemoFunction";

    public async void ExecuteLambda()
    {
        AmazonLambdaClient amazonLambdaClient = new
        AmazonLambdaClient(myCredentials, myRegionId);

        InvokeRequest invokeRequest = new InvokeRequest
        {
            FunctionName = myLambdaFunctionName,
            InvocationType = InvocationType.RequestResponse
        };

        InvokeResponse response = await
amazonLambdaClient.InvokeAsync(invokeRequest);

        if (response.StatusCode == 200)
        {
            Debug.Log("Successful lambda call!");
        }
    }
}
```

## 2.5 Sprint 5

### 2.5.1 Resumen

Al igual que como acostumbramos en otros *sprints*, dedicamos parte de los recursos a investigación y análisis de alternativas para llevar a la práctica distintas tareas. Este fue el caso de las políticas y roles que nos provee AWS y la API *gateway* que podría ser necesaria en un futuro. Planificamos comenzar a implementar la interfaz de una partida, con la tarea de diseño de código que conlleva. Por último, a demás de enfocarnos en generar un entorno de *test local*, realizamos tareas de creación de

librerías compartidas (*server/client*) y librería de *gameplay* basándonos en el diagrama de clases.

### 2.5.2 Tareas

Tareas a llevar a cabo en este *sprint*.

<b>Id</b>	<b>Título</b>	<b>Tiempo Estimado/ Real hs</b>	<b>Estado</b>
HOOD-74	Match Scene	6/4	Finalizada
HOOD-75	Código Compartido Entre Cliente y Servidor	2/2	Finalizada
HOOD-76	Clases de Gameplay	2/2	Finalizada
HOOD-77	GameLift local	4/4	En Progreso
HOOD-78	AWS Roles and Policies	2/8	Finalizada
HOOD-79	API Gateway con Lambda	1/1	Finalizada

Tabla 18. Tareas Sprint 5

#### 2.5.2.1 Reuniones

<b>Objetivo</b>	<b>Fecha</b>	<b>Participantes</b>	<b>Duración</b>
Pair programming UI	24 Jul 2022	Agustín, Manuel	3h
Sync pre entrega de	26 Jul 2022	Agustín, Ramiro, Manuel	2h

Tabla 19. Reuniones Sprint 5

### 2.5.3 Implementación - Match Scene

En este *sprint* se comenzó a implementar la *UI* de la escena de *matches*. Por la naturaleza de esta escena, que está ligada intrínsecamente al juego mismo, el avance de esta *UI* en este *sprint* no contiene mucha funcionalidad. La excepción siendo el tablero, que es creado dinámicamente, aunque sin interacción con el resto de los componentes del juego.

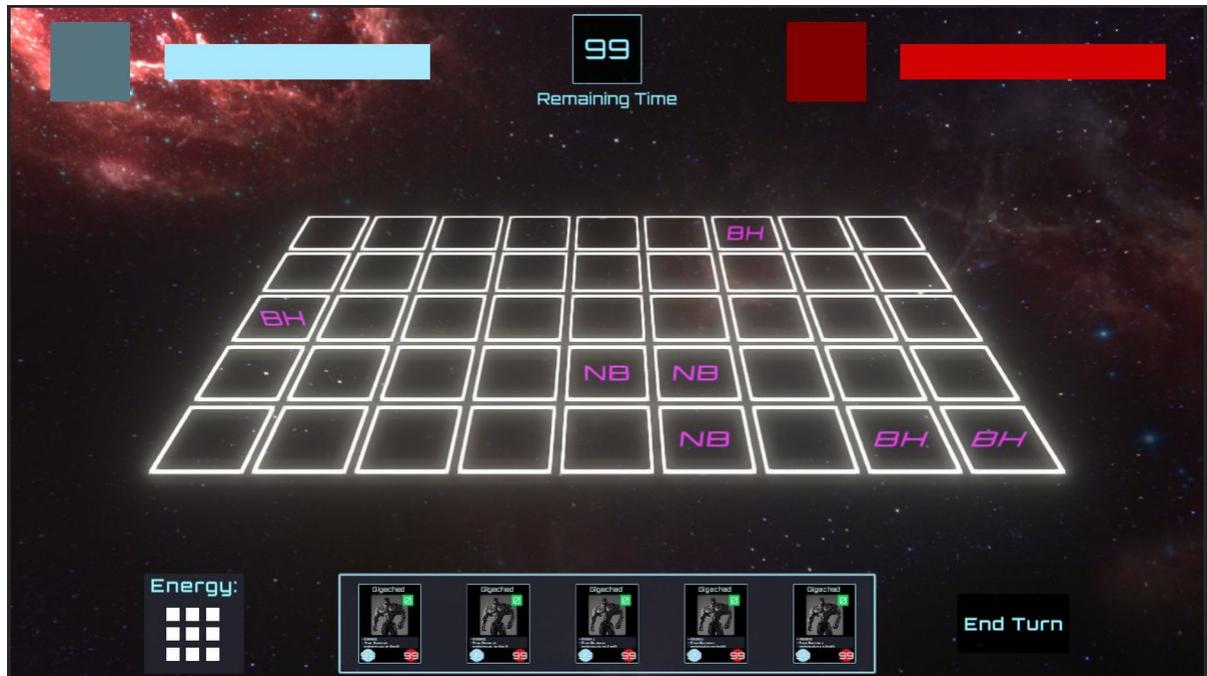


Figura 39. Escena de partida en juego

### 2.5.4 Implementación - Código Compartido Entre Cliente y Servidor

En una relación de cliente/servidor es muy importante que el servidor oficie muchas veces de validador de las acciones del cliente, es por esto que consideramos de vital importancia generar una librería en forma de *DLL* [g10] para disponer tanto en el código del servidor como en el código cliente. A esta librería le llamamos `SharedScripts`. Se encarga por ejemplo de almacenar los enumerados y los distintos mensajes de comunicación entre servidor y cliente en ambas direcciones. Es una manera de estandarizar estos mensajes y reducir la posibilidad de error humano. A su vez este agregado implicó generar un tercer *workspace* en PlasticSCM para el control de versión de la librería.

```

2 referencias
public enum ClientMessage
{
    UNDEFINED,
    CONNECT,
    KEY_PRESSED,
    REQUEST_PRIVATE_LOBBY,
    DISCONNECT_FROM_LOBBY,
    DISCONNECT_FROM_LOBBY_SECONDARY
}

2 referencias
public enum ServerMessage
{
    UNDEFINED,
    CONNECTED,
    START,
    CREATED_PRIVATE_MATCH,
    LOBBY_FULL,
    DISCONNECTED_FROM_LOBBY,
    WIN,
    LOSE
}

```

Figura 40. Mensajes entre servidor y cliente hasta el momento

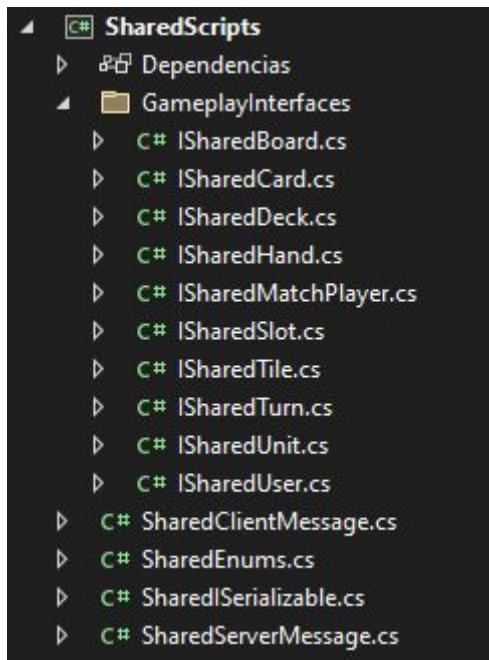


Figura 41. Vista explorador de solución de librería SharedScripts

### 2.5.5 Implementación - Clases de Gameplay

Para comenzar a desarrollar la parte de jugabilidad, debíamos bajar a código el diagrama UML de clases original. Es así que creamos un package llamado `Gameplay` dentro de la solución de cliente con las respectivas clases y relaciones entre ellas. De momento contamos con un `FakeGameManager` para manejar algunos estados del juego y probar distintas funcionalidades.

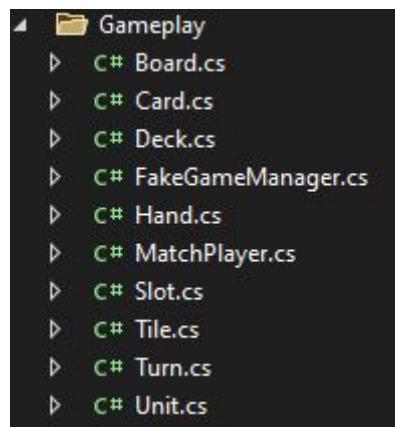


Figura 42. Package Gameplay

### 2.5.6 Implementación - GameLift local

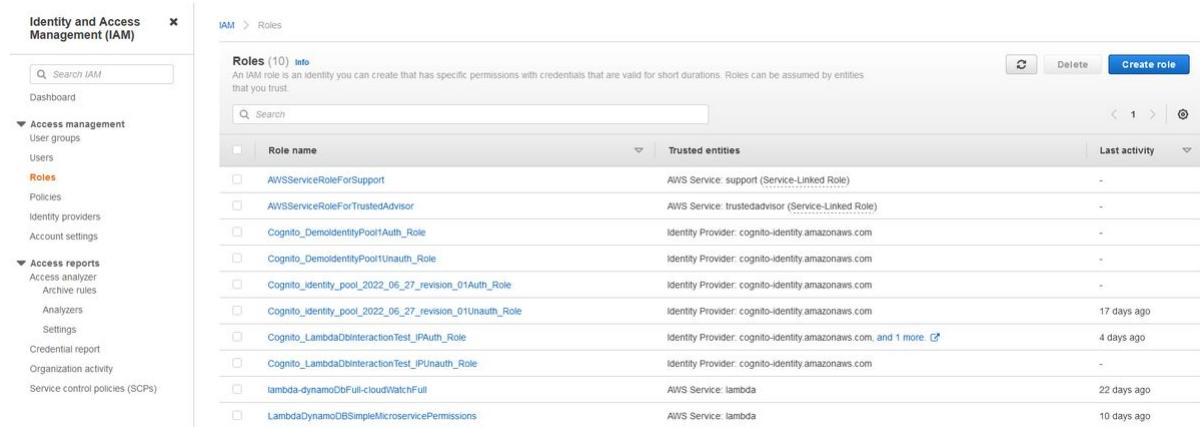
Desde un principio supimos que íbamos a necesitar un entorno local para poder probar funcionalidades con mayor agilidad. Esta necesidad fue haciéndose más notoria al momento de desarrollar y probar, por ejemplo, el *lobby*. Para llevar a la realidad esto, intentamos seguir una guía que proporciona Amazon. Estos pasos consisten en ejecutar mediante comando en la consola un archivo Java, proporcionándole como parámetro un puerto, luego ejecutar el servidor, ya sea desde Unity o habiendo generado un ejecutable, y por último, probar las funcionalidades desarrolladas desde un cliente. Podemos ver el estado del servidor en la consola y los *logs* de las distintas operaciones que realizamos. De momento no está terminado el desarrollo del entorno local debido a que hay ciertas operaciones como `SearchGameSession` que no están implementadas por Amazon en el archivo Java `GameLiftLocal.jar`.

### 2.5.7 Investigación - AWS Roles and Policies

Amazon gestiona acceso y utilización de sus servicios usando objetos llamados *policies*. Las *policies* pueden asociarse tanto a roles de usuario como a servicios. Es importante entender como se construye y funciona estos conceptos para poder administrar el uso de los servicios contratados y mantener un entorno seguro.

### 2.5.7.1 Roles

Los *roles* agrupan *users* de acuerdo a que se espera que sean capaces de hacer dentro de la infraestructura de la AWS Account. Algunos *roles* pueden ser: usuarios no autenticados, usuarios autenticados, administradores, moderadores, desarrolladores. Se puede acceder, editar o crear *roles* disponibles para la AWS Account a través del servicio de Amazon IAM.



The screenshot shows the AWS Identity and Access Management (IAM) service interface. On the left, there's a navigation sidebar with options like 'Identity and Access Management (IAM)', 'Dashboard', 'Access management' (with 'Roles' selected), 'Policies', 'Identity providers', and 'Account settings'. The main content area is titled 'Roles (10) Info' and contains a table with 10 rows, each representing a role. The columns in the table are 'Role name', 'Trusted entities', and 'Last activity'. The roles listed are: 'AVSServiceRoleForSupport', 'AVSServiceRoleForTrustedAdvisor', 'Cognito\_DemoIdentityPool1Auth\_Role', 'Cognito\_DemoIdentityPool1Unauth\_Role', 'Cognito\_identity\_pool\_2022\_06\_27\_revision\_01Auth\_Role', 'Cognito\_identity\_pool\_2022\_06\_27\_revision\_01Unauth\_Role', 'Cognito\_LambdaDbInteractionTest\_IAuth\_Role', 'Cognito\_LambdaDbInteractionTest\_IPUnauth\_Role', 'lambda-dynamoDbFullCloudWatchFull', and 'LambdaDynamoDBSimpleMicroservicePermissions'. The 'Last activity' column shows dates ranging from 17 days ago to 22 days ago.

Figura 43. Roles en plataforma de AWS

### 2.5.7.2 Policies

Documentos de texto en formato JSON que describen servicios de AWS y que acciones se pueden realizar en esos servicios. Las *policies* se adjuntan a *roles*. Cualquier *user* con ese *role* puede acceder a los servicios descriptos en la *policy* y ejecutar esas acciones. Al igual que con los roles, se puede acceder, editar o crear *policies* disponibles para la AWS Account a través del servicio de Amazon IAM.

La siguiente *policy* permite invocar la función Lambda de nombre

`LambdaDbInteractionTest_L_2` asociada a la AWS Account de un Id específico (que fue tachado con rojo) en la `Region sa-east-1`. También permite ejecutar varias acciones de Gamelift a todos los recursos de Gamelift definidos en esta AWS Account.

The screenshot shows the AWS IAM Policies Summary page. On the left, there's a sidebar with navigation links like Dashboard, Access management, Policies (which is selected), and others. The main area shows a policy named 'LambdaDbInteractionTest\_P' with its ARN and a description. Below that is a tabbed interface with 'Permissions' selected, followed by Policy usage, Tags, Policy versions, and Access Advisor. Under Permissions, there are tabs for Policy summary, JSON (selected), and Edit policy. The JSON code is displayed:

```
1- {
2-   "Version": "2012-10-17",
3-   "Statement": [
4-     {
5-       "Sid": "AuthUsersLambdaInvokeAccess",
6-       "Effect": "Allow",
7-       "Action": "lambda:InvokeFunction",
8-       "Resource": "arn:aws:lambda:sa-east-1:XXXXXXXXXXXX:function:LambdaDbInteractionTest_L_2"
9-     },
10-    {
11-      "Sid": "PlayerPermissionsForManualGameSessions",
12-      "Effect": "Allow",
13-      "Action": [
14-        "gamelift>CreateGameSession",
15-        "gamelift:DescribeGameSessions",
16-        "gamelift:SearchGameSessions",
17-        "gamelift>CreatePlayerSession",
18-        "gamelift>CreatePlayerSessions",
19-        "gamelift:DescribePlayerSessions"
20-      ],
21-      "Resource": "*"
22-    }
23-  ]
24-}
```

Figura 44. Ejemplo de Policy en plataforma de AWS

Cada *policy* puede tener múltiples entradas llamadas *statements*. Los *statements* son útiles para agrupar los servicios que queremos habilitar de una manera legible. Cada *statement* tiene los siguientes componentes.

### Sid

Un identificador único dentro de cada *policy*, no tiene restricciones mínimas de caracteres ni requerimientos especiales mas allá de su unicidad dentro de esta *policy*. Por ejemplo:

```
"Sid":"LambdaInvokePermission",
```

### Effect

Que permiso define la *policy*. Todos los recursos están bloqueados por defecto. Los valores posibles son `allow` y `deny`.

```
"Effect":"Allow",
```

## Action

A que acciones realizables se aplica la *policy*, se puede definir una acción específica dentro de un AWS. Como por ejemplo dentro del servicio Lambda, la acción de invocar una función:

```
"Action": "Lambda:InvokeFunction",
```

## Resource

Selecciona las instancias específicas de los AWS donde es aplicable esta *policy*. Se puede restringir su aplicabilidad por región, AWS Account, instancia específica de un servicio, etc. Por ejemplo:

```
"Resource": "arn:aws:lambda:sa-east-1:123456789123:function:MyLambdaFunctionName"
```

Restringir la acción a el servicio Lambda, en la `Region sa-east-1`, para las instancias de Lambda definidas por el usuario 123456789123 y la función específica con el nombre `MyLambdaFunctionName`.

## Wildcards

Se puede usar “\*” como *wildcard*, lo que representa que todas las opciones posibles para ese atributo están seleccionadas. Por ejemplo:

```
"Resource": "arn:aws:lambda:sa-east-1:123456789123:function:/*"
```

Significa que todas las funciones lambda definidas por el usuario 123456789123 en la `Region sa-east-1` están seleccionadas para esta *policy*.

```
"Action": "Lambda:/*"
```

Significa que todas las acciones del servicio **Lambda** están seleccionadas para esta *policy*.

## Policy Simulator

Se puede usar un simulador de *policies* para evaluar tentativamente si una *policy* está estructurada apropiadamente para los servicios que se desea acceder. El simulador de *policies* no tiene acceso a información de la AWS Account por lo que selectores dentro del componente de Resource como identificador de usuario o nombres de recursos específicos creados bajo una AWS Account no serán evaluados correctamente y deben reemplazarse por *wildcards*.

The screenshot shows the IAM Policy Simulator interface. On the left, there is a sidebar titled "Policies" with a warning message: "Editing policy: LambdaDbInteractionTest\_P". The main area displays a JSON policy document:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AuthUsersLambdaInvokeAccess",
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Resource": "*"
    }
  ]
}
```

Below the policy, there are tabs for "Global Settings" and "Action Settings and Results". The "Action Settings and Results" tab shows a table of actions and their permissions. The table includes columns for Service, Action, Resource Type, Simulation Resource, and Permission. Most actions result in a "denied" permission, except for one "InvokeFunction" action which is marked as "allowed".

Service	Action	Resource Type	Simulation Resource	Permission
AWS Lambda	GetFunctionUrlConfig	function	*	denied Implicitly denied (no matching state...)
AWS Lambda	GetLayerVersion	layerVersion	*	denied Implicitly denied (no matching state...)
AWS Lambda	GetLayerVersionPolicy	layerVersion	*	denied Implicitly denied (no matching state...)
AWS Lambda	GetPolicy	function	*	denied Implicitly denied (no matching state...)
AWS Lambda	GetProvisionedConcurrencyConfig	not required	*	denied Implicitly denied (no matching state...)
AWS Lambda	InvokeAsync	function	*	denied Implicitly denied (no matching state...)
AWS Lambda	InvokeFunction	function	*	allowed 1 matching statements
AWS Lambda	InvokeFunctionUrl	function	*	denied Implicitly denied (no matching state...)
AWS Lambda	ListAliases	function	*	denied Implicitly denied (no matching state...)
AWS Lambda	ListCodeSigningConfigs	not required	*	denied Implicitly denied (no matching state...)
AWS Lambda	ListEventSourceMappings	not required	*	denied Implicitly denied (no matching state...)
AWS Lambda	ListFunctionEventInvokeConfigs	function	*	denied Implicitly denied (no matching state...)
AWS Lambda	ListFunctionUrlConfigs	function	*	denied Implicitly denied (no matching state...)
AWS Lambda	ListFunctions	not required	*	denied Implicitly denied (no matching state...)
AWS Lambda	ListFunctionsByCodeSigningConfig	code signing config	*	denied Implicitly denied (no matching state...)

Figura 45. Simulador de Policy en plataforma AWS

## 2.5.8 Investigación - API Gateway con Lambda

Como mencionamos anteriormente se puede interactuar con Lambda desde dentro de código de Unity, pero pueden ocurrir situaciones donde necesitamos interactuar con nuestros servicios como funciones Lambda incluso antes de recorrer el login o siquiera abrir el juego. Para eso AWS provee el servicio API Gateway que cumple la función de un punto de entrada público con el que podemos interactuar desde cualquier estado.

Agregando “Amazon.Lambda.APIGatewayEvents” a través de *NuGet package manager* en Visual Studio a nuestro proyecto de Lambda, podemos expandir el tipo de dato que recibe y devuelve nuestra función Lambda para que conforme con los protocolos de comunicación HTTP.

```
using Amazon.Lambda.Core;
using Amazon.Lambda.APIGatewayEvents;
```

```

// Assembly attribute to enable the Lambda function's JSON input to be
converted into a .NET class.
[assembly:
LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambd
aJsonSerializer))]

namespace DotNETDemoFunction;

public class Function
{

    /// <summary>
    /// A simple function that takes a string and does a ToUpper
    /// </summary>
    /// <param name="input"></param>
    /// <param name="context"></param>
    /// <returns></returns>
    public APIGatewayProxyResponse FunctionHandler(APIGatewayProxyRequest
request, ILambdaContext context)
{
    // Early exit if no parameters were received
    if (request.QueryStringParameters == null)
    {
        return new APIGatewayProxyResponse
        {
            StatusCode = 400,
            Body = $"Bad Request: request.QueryStringParameters == null"
        };
    }

    // Return value
    string name = "No Name";

    if (request.QueryStringParameters.ContainsKey("name"))
    {
        name = request.QueryStringParameters["name"];
    }

    // Outputs to aws cloud watch logs
    context.Logger.Log($"Got name: {name}");

    // Do something with your parameter here
    name.ToUpper();

    // Successful response to client
    return new APIGatewayProxyResponse
    {
        StatusCode = 200,
        Body = $"User name received: {name}"
    };
}

```

}

Una vez que tenemos definida nuestra nueva función se puede activar un nuevo API Gateway desde la interfaz web de AWS.

The screenshot shows the AWS Management Console interface for creating a new API. At the top, the AWS logo and services like Cognito, IAM, Amazon GameLift, Lambda, and DynamoDB are visible. The search bar contains 'Search for services, features, blogs, docs, and more'. The location is set to 'São Paulo' and the user is 'gmail.com'. Below the header, the 'Amazon API Gateway' service is selected. The main navigation bar shows 'APIs > Create'. On the left, a section titled 'Choose the protocol' offers 'REST' (selected) and 'WebSocket' options. A note explains that a REST API is a collection of resources and methods. Below this, three radio button options are shown: 'New API' (selected), 'Import from Swagger or Open API 3', and 'Example API'. The 'Settings' section allows defining the API name ('DotNET-test-api'), adding a description, and selecting the 'Endpoint Type' as 'Regional'. A note indicates that the 'API name\*' field is required. A blue 'Create API' button is at the bottom right.

Figura 46. Interfaz de activación de API Gateway en AWS

Y la podemos integrar a nuestra función Lambda

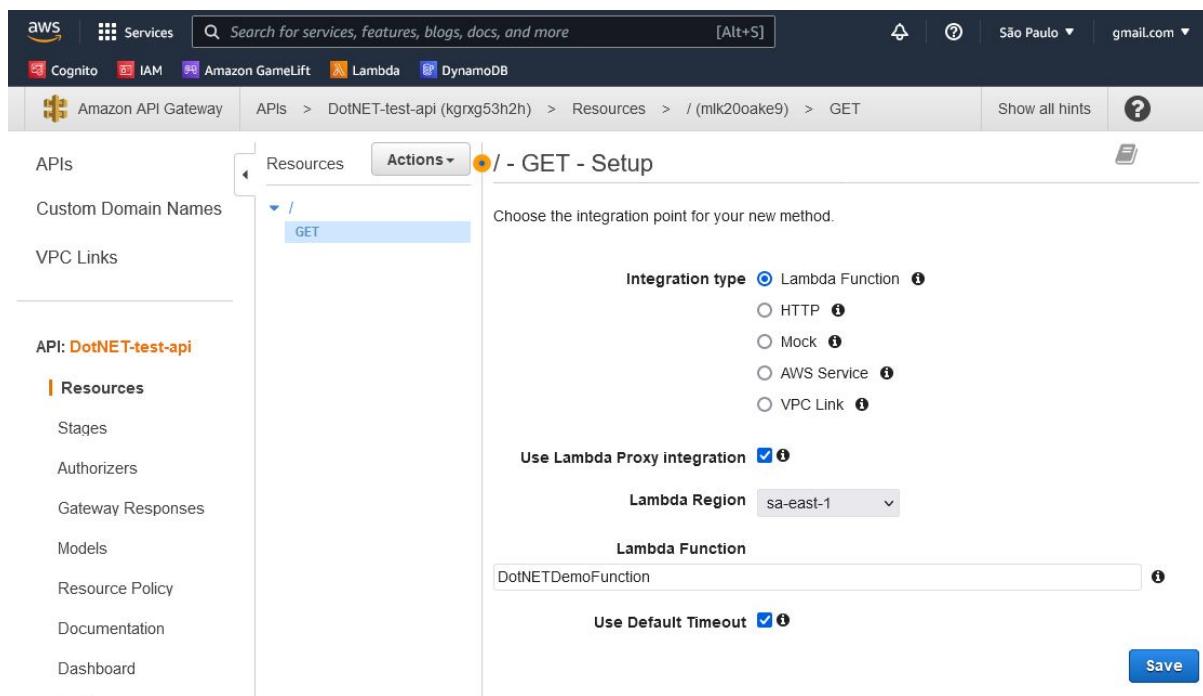


Figura 47. Interfaz para integración de función lambda en AWS

AWS creará una URL para nuestra API que podemos probar rápidamente agregando parámetros y ejecutando en cualquier navegador como por ejemplo de la siguiente manera.

```
https://kgrxg53h2h.execute-api.sa-east-1.amazonaws.com/prod?name=Test
```

Lo cual devuelve una respuesta exitosa.

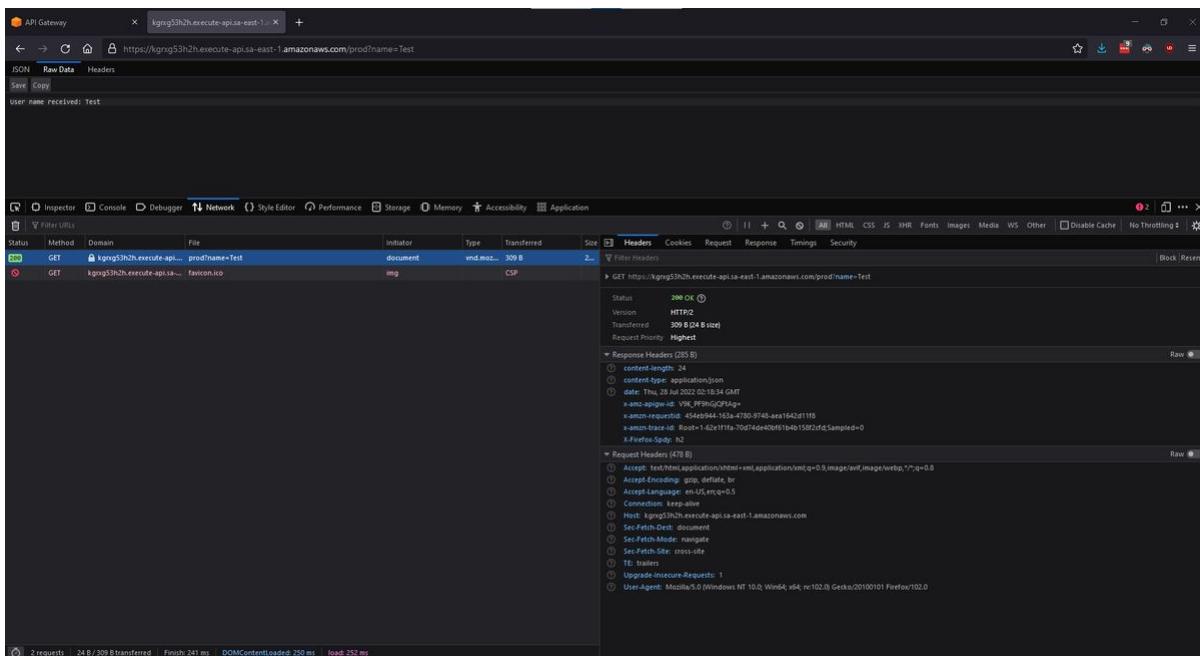


Figura 48. Respuesta observada de API de AWS desde un navegador

## 2.6 Sprint 6

### 2.6.1 Resumen

En este sprint planeamos diseñar partes esenciales para el proyecto como lo son la información estática y la instanciación de objetos en el juego. Es esencial avanzar con el tema referente al *testing* local para agilitar el desarrollo. También debemos hacer trabajos más administrativos como la creación de una cuenta de AWS para el grupo con sus respectivos usuarios, evaluar el estado del proyecto, asegurarnos que todos estén actualizados en el repositorio y analizar el *sprint* siguiente. A su vez se empezó a trabajar en la lógica interna del *gameplay* del juego.

### 2.6.2 Tareas

Tareas a llevar a cabo en este *sprint*.

<b>Id</b>	<b>Título</b>	<b>Tiempo Estimado/ Real hs</b>	<b>Estado</b>
HOOD-77	GameLift Local	6/14	CANCELADA

HOOD-80	Lógica interna del tablero	6/7	FINALIZADA
HOOD-81	Lógica interna de animaciones	4/4	FINALIZADA

Tabla 20. Tareas Sprint 6

### 2.6.2.1 Reuniones

Objetivo	Fecha	Participantes	Duración
• Diseño de data estática e instanciación de objetos in game	02 Aug 2022	Agustín, Manuel	2h
• Creación de AWS Account y usuarios administradores	03 Aug 2022	Agustín, Ramiro, Manuel	1.5h
• Evaluación del estado del proyecto • Sprint planning • Diseño interacción de gameplay	04 Aug 2022	Agustín, Ramiro, Manuel	1.5h
• InGameObjectfactory/Pool review	06 Aug 2022	Agustín, Manuel	2h
• Team sync and fixing SharedScripts merge issues between repositories	07 Aug 2022	Agustín, Ramiro, Manuel	3h
• Team sync	10 Aug 2022	Agustín, Ramiro, Manuel	2h

Tabla 21. Reuniones Sprint 6

### 2.6.3 Re-Evaluación de GameLift Local

Luego de acarrear la tarea referida al entorno de *testing* local utilizando GameLift Local entre *sprints*, tomamos la decisión de cancelar esta tarea para enfocarnos en realizar progreso por otro camino.

Las razones por considerar GameLift Local en un primer lugar se pueden resumir de la siguiente forma:

- Posibilidad de levantar un servidor desde Unity eliminando tiempo en creación de servidor
- Agilidad en detección de errores
- Posibilidad de debug para analizar casos de errores

- Eliminar el tiempo empleado en creación de fleets (30 min. en eliminar y crear una nueva)

Si bien identificamos los anteriores beneficios de poder utilizar GameLift Local, el tiempo consumido por este esfuerzo es potencialmente mayor a la ganancia esperada y no creemos que exista un *workaround* para obtener lo que esperábamos. Consideramos que la parte de conexión con GameLift en nuestro proyecto se encuentra bien integrada y podemos prescindir de un entorno de *testing* local de las *features* de GameLift. Igualmente es imprescindible contar en un futuro con un entorno de testing local que evite GameLift para probar funcionalidades de *gameplay*.

#### 2.6.4 Consolidación de AWS Account

Cada cuenta de AWS agrupa y relaciona los servicios que el desarrollador consume. Realizar un proceso de autenticación de Cognito de una AWS Account, nos identifica como miembros de una User Pool definida en la misma AWS Account que tiene asignando un IAM rol creado bajo la misma AWS Account y así sucesivamente.

Hasta ahora, en busca de agilidad e independencia, estábamos usando recursos de amazon mediante cuentas personales, eso traía algunos inconvenientes a la hora de consolidar el proyecto ya que identificadores y roles de una AWS Account no están presentes en otras.

En este punto en el desarrollo decidimos consolidar todos los usuarios bajo una misma AWS Account.

Amazon organiza sus usuarios de manera que dependiendo del rol que asumen, se habilitan diferentes funcionalidades. En nuestro caso generamos una estructura de 3 niveles con diferentes permisos.

- Root: Acceso a editar administradores.
- Administradores: Acceso a definir y editar servicios de Amazon usados por la misma AWS Account
- Usuarios autenticados: Acceso a limitados servicios de Amazon referidos a la funcionalidad del juego como crear cuentas de usuario, iniciar juegos o unirse a juegos existentes.

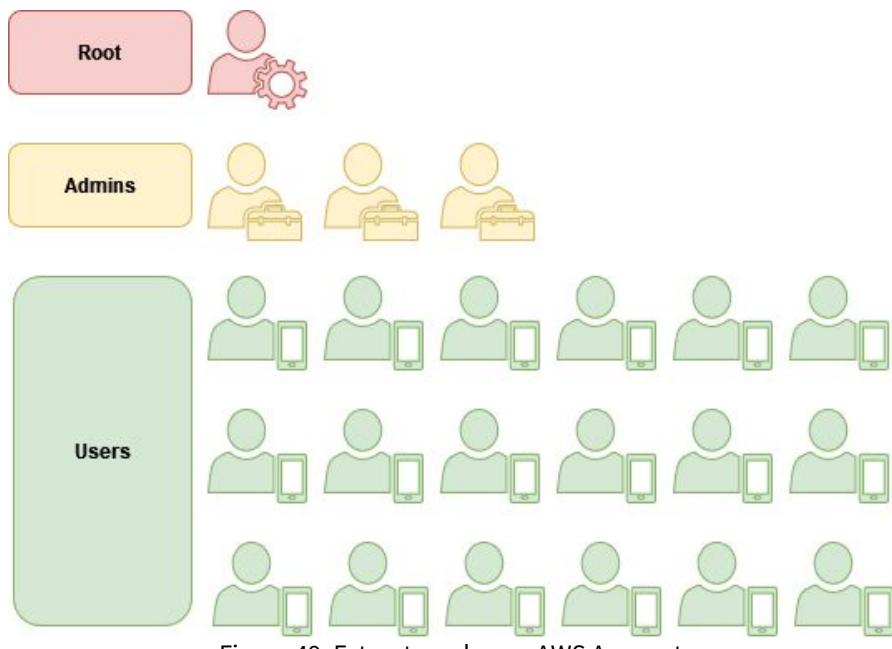


Figura 49. Estructura de una AWS Account

## 2.6.5 Lógica Auxiliar del Tablero

Para facilitar la lógica necesaria para implementar las acciones que son posibles en el juego, fue necesaria la creación de métodos auxiliares que permiten devolver “formas” representadas por celdas en el tablero.

Estas formas son: Cuadrado, Cruz y Línea.

Todas estas formas son conseguidas por un método público

`GetShapeFromCenterTileCoord`, que tomando en cuenta una celda central, un tamaño y si incluye su centro, devuelve una lista de celdas que dibujan la forma indicada por un *enum*.

### 2.6.5.1 Cuadrado

```

private List<SharedTile> GetSquareFromCentralCoord(int shapeSize,
Vector2Int centerTileCoord, bool includeCenter)
{
    List<SharedTile> tiles = new List<SharedTile>();
    for (int x = centerTileCoord.x - shapeSize; x <= centerTileCoord.x +
shapeSize; x++)
    {
        for (int y = centerTileCoord.y - shapeSize; y <= centerTileCoord.y
+ shapeSize; y++)
        {
    }
}

```

```

        if (x != centerTileCoord.x || y != centerTileCoord.y)
    {
        SharedTile tileToAdd = GetTile(x, y);
        if (tileToAdd != null)
        {
            tiles.Add(tileToAdd);
        }
    }
}
if (includeCenter)
{
    tiles.Add(GetTile(centerTileCoord.x, centerTileCoord.y));
}

return tiles;
}

```

### 2.6.5.2 Cruz

La cruz toma un rol más prominente en el proyecto, ya que puede usarse para conseguir las celdas adyacentes a una celda central, dibujando una cruz de tamaño 1 y sin centro.

```

private List<SharedTile> GetCrossFromCentralCoord(int shapeSize, Vector2Int
centerTileCoord, bool includeCenter)
{
    List<SharedTile> tiles = new List<SharedTile>();
    for (int x = centerTileCoord.x - shapeSize; x <= centerTileCoord.x +
shapeSize; x++)
    {
        for (int y = centerTileCoord.y - shapeSize; y <= centerTileCoord.y
+ shapeSize; y++)
        {
            if (x == centerTileCoord.x || y == centerTileCoord.y)
            {
                SharedTile tileToAdd = GetTile(x, y);
                if (tileToAdd != null)
                {
                    tiles.Add(tileToAdd);
                }
            }
        }
    }
    if (!includeCenter)
    {
        tiles.Remove(GetTile(centerTileCoord.x, centerTileCoord.y));
    }
}

```

```
        return tiles;
    }
```

### 2.6.5.3 Línea

A diferencia de las anteriores, al no ser simétrica del centro. La línea necesita una coordenada adicional para indicarle en qué dirección es dibujada.

```
private List<SharedTile> GetLineFromCentralCoord(int shapeSize, Vector2Int centerTileCoord, Vector2Int directionTileCoord, bool includeCenter)
{
    List<SharedTile> tiles = new List<SharedTile>();

    if (directionTileCoord == null)
    {
        return tiles;
    }

    Vector2Int netDirectionVector = centerTileCoord - directionTileCoord;

    if (netDirectionVector.x < 0) // Rightwards direction
    {
        for (int x = centerTileCoord.x; x <= centerTileCoord.x + shapeSize;
x++)
        {
            SharedTile tileToAdd = GetTile(x, centerTileCoord.y);
            if (tileToAdd != null)
            {
                tiles.Add(tileToAdd);
            }
        }
    }
    else if (netDirectionVector.x > 0) // Leftwards direction
    {
        for (int x = centerTileCoord.x; x >= centerTileCoord.x - shapeSize;
x--)
        {
            SharedTile tileToAdd = GetTile(x, centerTileCoord.y);
            if (tileToAdd != null)
            {
                tiles.Add(tileToAdd);
            }
        }
    }
    else if (netDirectionVector.y < 0) // Upwards direction
    {
        for (int y = centerTileCoord.y; y <= centerTileCoord.y + shapeSize;
y++)
    }
```

```

    {
        SharedTile tileToAdd = GetTile(centerTileCoord.x, y);
        if (tileToAdd != null)
        {
            tiles.Add(tileToAdd);
        }
    }
    else if (netDirectionVector.y > 0) // Downwards direction
    {
        for (int y = centerTileCoord.y; y >= centerTileCoord.y - shapeSize;
y--)
        {
            SharedTile tileToAdd = GetTile(centerTileCoord.x, y);
            if (tileToAdd != null)
            {
                tiles.Add(tileToAdd);
            }
        }
    }

    if (!includeCenter)
    {
        tiles.Remove(GetTile(centerTileCoord.x, centerTileCoord.y));
    }

    return tiles;
}

```

## 2.6.6 Lógica de Animaciones

A pesar de que una representación visual fidedigna no es el enfoque del proyecto, igual consideramos que debía ser necesario construir una infraestructura sólida y escalable que permita una fácil implementación de las animaciones de las unidades.

Unity posee un sistema interno para manejar animaciones llamado “Mecanim” [v21] , que permite controlar el flujo de éstas mediante una máquina de estados. Esto puede ser útil, aunque complejo, para proyectos donde el flujo de animaciones debe ser en tiempo real y con altos grados de variantes.

En nuestro caso, notamos que las animaciones siempre tendrían el mismo flujo: Acción → *Idle* (Reposo). Lo que nos llevó a la siguiente solución:

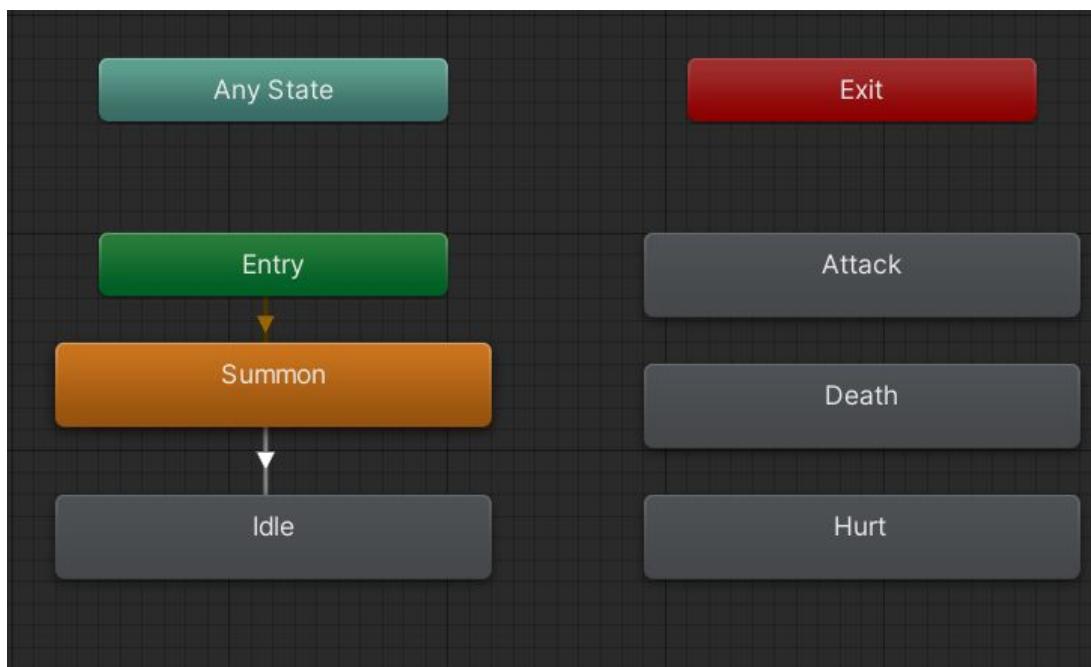


Figura 50. Interfaz de usuario de la máquina de estados del animador “Mecanim”.

En el diagrama de arriba se observan lo siguiente:

Cada *gameObject* animable tiene un componente llamado *Animator Controller* [v22]. Este componente controla las animaciones de ese objeto guardándolas e identificándolas mediante un *string* idéntico al nombre de su estado, representados por los rectángulos más grandes — en gris y anaranjado. Este *string* puede convertirse en un *hash* para facilitar su uso.

El estado *summon* tiene flechas entrando y saliendo de él. Esto representa que al crearse una unidad en el juego — *entry* — el juego automáticamente pasará a reproducir la animación de *summon*, y después de terminada ésta, se pasará a la animación de *idle*.

El resto de las animaciones no poseen flechas, ya que se reproducen por código. El método que reproduce estas animaciones se llama en cualquier momento que deseemos reproducir una animación.

```
public async Task PerformAnimation(int anAnimation)
{
    float duration = GetAnimationLength(anAnimation);
    if (duration != -1)
    {
        myAnimator.CrossFade(anAnimation, 0);
        float end = Time.time + duration;

        while (Time.time < end)
        {
            await Task.Yield();
        }
    }
}
```

```

        }
        myAnimator.CrossFade(SharedUnit.IdleAnimation, 0);
    }
}

```

Esto no es suficiente para hacer un sistema de animaciones, sin embargo. Si simplemente usamos el mismo *Animator Controller* para todas las unidades, todas las unidades reproducirán el mismo set de animaciones y todas se verán idénticas entre sí. Por esta razón, por cada tipo de unidad necesita un *Animator Override Controller* [v23], que hereda todas sus animaciones de un *Animator Controller* base y permite sobre-escribir las conservando su nombre.

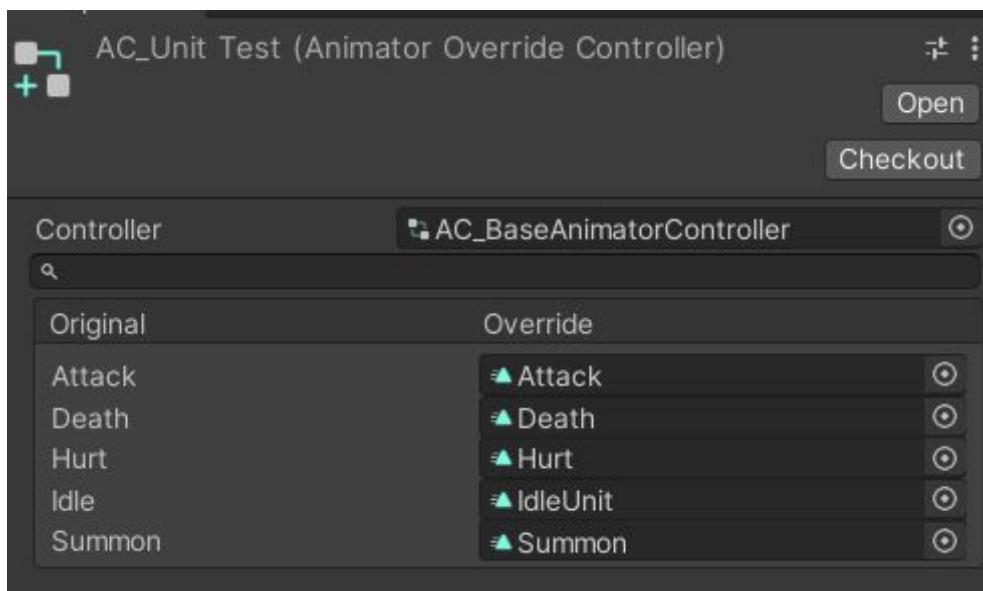


Figura 51. Interfaz del Animator Override Controller.

En nuestro caso, por razones de tiempo, las únicas animaciones que se sobre-escribieron fueron las animaciones de reposo. Esto se debe a que ésta es la animación que se reproduce cuando las unidades están en el tablero. Esto permite al jugador diferenciarlas mientras juega.

Entonces, al crearse una nueva unidad, ésta se asigna su *Animator Override Controller* correspondiente y así puede acceder a las animaciones correctas.

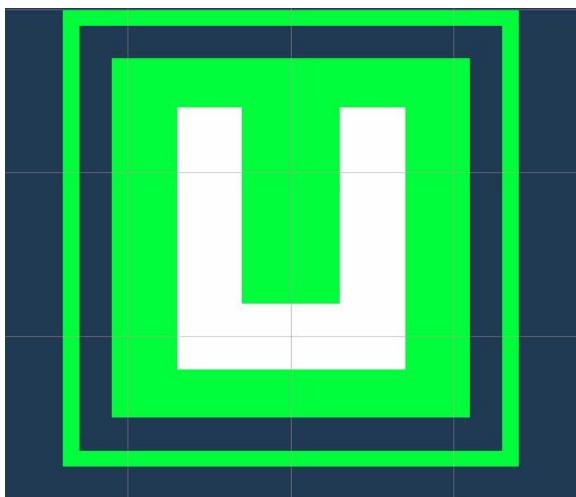


Figura 52. Sprite de TestUnit

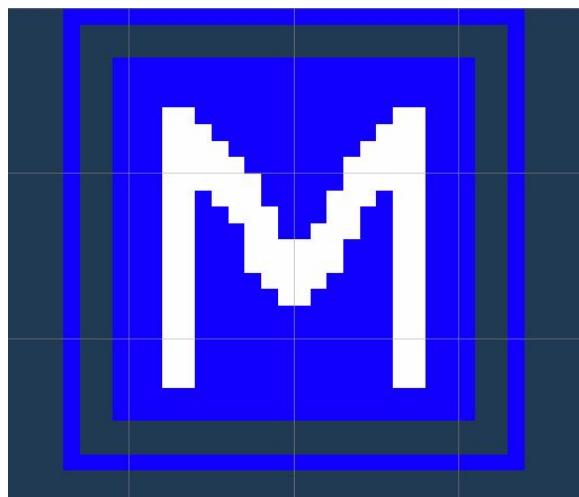


Figura 53. Sprite de Test Mothership

## 2.7 Finalización 1

### 2.7.1 Re-Estructuración de Ambiente de Desarrollo

**Duración estimada:** 6d

Como fue mencionado previamente, nuestro entorno de desarrollo actual presentaba algunos problemas de ergonomía, en vista de esto y conociendo mas íntimamente las necesidades del proyecto, decidimos reconstruir el repositorio teniendo los siguientes requerimientos.

#### 2.7.1.1 Un Repositorio

El entorno anterior estaba compuesto de tres repositorios de Plastic SCM; Client, Server y Shared. Esto provocaba inconvenientes ya que los miembros del equipo necesitaban mantener sus tres repositorios actualizados para no desfasarse, lo que en ocasiones incurría en gastos de tiempo a la hora de resolver conflictos de consolidación.

En su lugar todo el proyecto debe estar contenido en un solo repositorio para facilitar procesos de consolidación de código.

#### 2.7.1.2 Una Solución

El entorno anterior estaba compuesto de tres soluciones de Visual Studio; Client, Server y Shared.

Esto provocaba inconvenientes al tener que copiar DLL's cuando necesitábamos referenciar clases entre soluciones. Tanto Unity como Plastic presentaron errores al tratar de incorporar código externo de esta manera.

En su lugar todo el código debe estar definido por una solución de Visual Studio.

### 2.7.1.3 Un Proyecto de Unity

El entorno anterior estaba compuesto de dos proyectos de Unity; Client y Server. Esto nos permitía simplificar los pasos a la hora de generar versiones compiladas del proyecto ya que el proyecto de Client solo debía encargarse de compilar las escenas del juego en un ejecutable de Windows. Mientras que el proyecto de Server debía encargarse la escena única de server en un ejecutable de Linux.

Esto provocaba inconvenientes ya que si queríamos representar el mismo objeto en los dos proyectos era necesario exportar paquetes de Unity e importarlos en el proyecto objetivo, un proceso que debía repetirse con cada actualización a cada objeto.

En su lugar todas las escenas y objetos deben existir y ser accesibles desde el mismo proyecto.

### 2.7.1.4 Ejecución

Este proceso implica importar y unir código de tres proyectos por lo que consumió una cantidad de tiempo considerable y se desarrolla a lo largo de dos grandes etapas. En la etapa 1 se genera un proyecto con código resultante de la unión de los tres proyectos anteriores en el nuevo ambiente de desarrollo y ocurre en paralelo a la tarea de desarrollo de *gameplay*, la etapa 2 integra el nuevo código de *gameplay* con el nuevo ambiente de desarrollo.



Figura 54. Línea de tiempo, re-estructuración de ambiente de desarrollo.

## 2.7.2 Otras Tareas de Refactorización

También en esta primera etapa se llevó a cabo un proceso de normalización de datos y métodos de log.

### 2.7.2.1 Normalización de datos constantes

**Duración estimada:** 4d

Las entidades de *gameplay* deben ser parametrizables, podemos tener muchas unidades pero todas deberían tener los mismos atributos básicos de salud total, velocidad, poder. Unity provee el concepto de Scriptable Object para este fin. Un Scriptable Object es una abstracción del lenguaje que permite definir una clase con atributos públicos, luego usando el editor generar muchas instancias de esa clase dejando la opción al desarrollador de elegir valores diferentes para los atributos de esas instancias.

Elegimos asignar un identificador único basado en *enums* a cada instancia de un scriptable object de manera que sólo pasando este identificador cualquier aplicación, ya sea cliente o servidor, puede buscar el resto de los datos de esa instancia.

Por lo tanto cada dato constante se forma de 3 partes. Una definición de la clase, una lista de instancias de esa clase, un identificador único para cada ítem de esa lista. El siguiente es un ejemplo basado en nuestra entidad de *gameplay* de Unit.

#### Definición de Units

```
using System.Collections.Generic;
using UnityEngine;
using SharedScripts.DataId;

[System.Serializable]
public class UnitData
{
    public string myName;
    public UnitId myId;

    public int myAttack;
    public int myAttackRange;
    public int myShields;
    public int myMovementRange;
    public AbilityId myAbilityId;

    public bool canSpawnOtherUnits;
```

```
    public AnimatorOverrideController myOverrideAnimatorController;
    public Sprite mySprite;
}

[CreateAssetMenu(fileName = "Units_Inst", menuName = "DataListsInstances/
Units_Inst")]
public class Units_Def : ScriptableObject
{
    [SerializeField] public List<UnitData> myUnits;
}
```

## Lista de Units

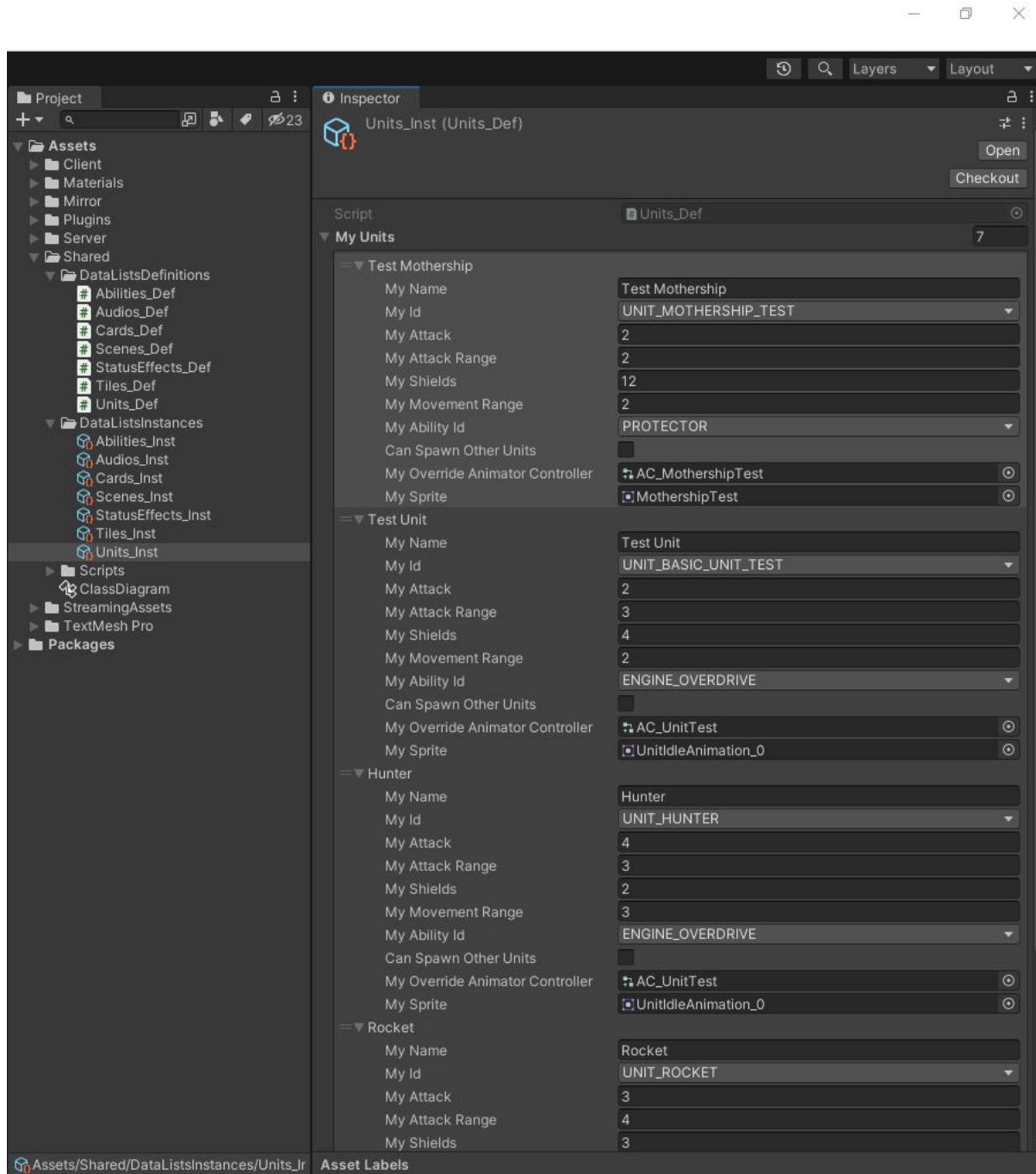


Figura 55. Lista de instancias de la clase UnitData

Es de valor notar la relación 1-1 de los atributos en la definición de la clase, con los campos editables que podemos ver en el editor.

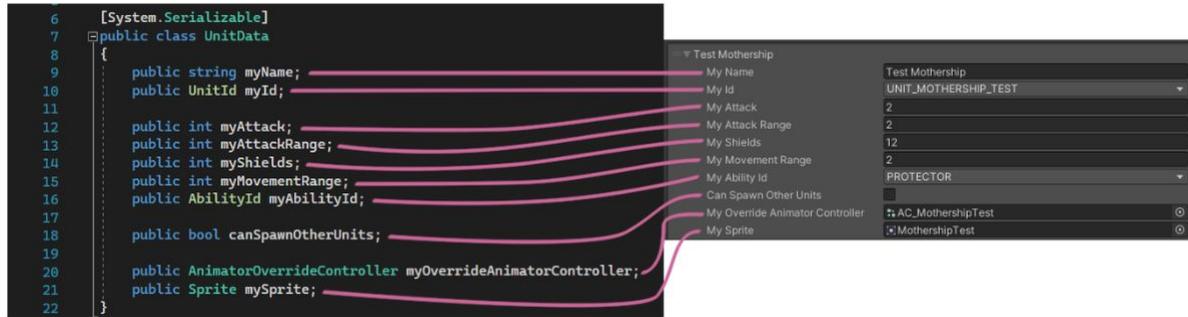


Figura 56. Mapeo de atributos a campos editables.

## Identificador de Units

Elegimos *enums* como id's ya que proveen listas separadas que aseguran unicidad y son fácilmente legibles. Adicionalmente la interfaz de Unity convierte cualquier campo basado en *enum* en un *dropdown*, reduciendo la posibilidad de que un desarrollador ingrese un *id* equivocado.

```
namespace SharedScripts
{
//...
    namespace DataId
    {
        public enum UnitId
        {
            INVALID,
            UNIT_MOTHERSHIP_TEST,
            UNIT_BASIC_UNIT_TEST,
            UNIT_3,
            UNIT_HUNTER,
            UNIT_ROCKET,
            UNIT_TANK,
            UNIT_FENIX,
            UNIT_TRAVELER
        }
    }
}
```

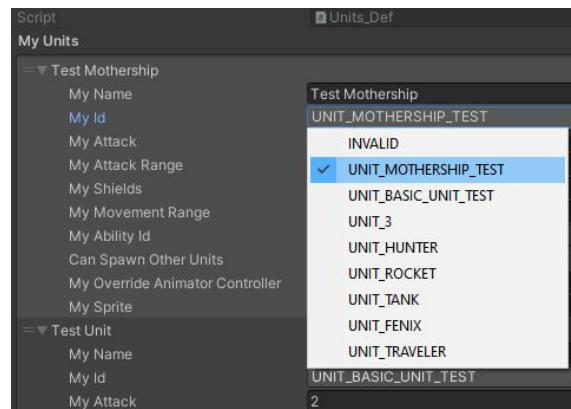


Figura 57. Selección de identificador en un Scriptable Object

### 2.7.2.2 Carga de Datos

Así como definimos los datos, también debemos cargarlos, esto nos da una oportunidad de derivar todas las tareas de carga al tiempo donde el usuario espera que hayan

demoras, durante el inicio de la aplicación y de cerrar la aplicación si un dato que fue marcado como esperado no es cargado exitosamente. Sistematizamos este proceso en una clase llamada `SharedDataLoader` que ejecuta el siguiente patrón.

```
public class SharedDataLoader : MonoBehaviour
{
    //...

    [SerializeField] private Units_Def myUnitsData; // referencia a la lista de
    instancias
    [SerializeField] private bool myIsRequiredUnitsData; // marcador de dato
    esperado
    private Dictionary<UnitId, UnitData> myUnits; // el dato cargado en memoria

    private Dictionary<string, bool> myRequiredDatasMap; // mapa de datos
    esperados
    private Dictionary<string, bool> myLoadedDatasMap; // mapa de datos
    cargados exitosamente

    // usa los marcadores myIsRequiredUnitsData para popular myRequiredDatasMap
    private void GetDataTargetInfo(){}

    // llama a metodos de carga especificos para cada lista si
    // myRequiredDatasMap lo indica
    private void LoadTargetData(){}

    // carga los datos de myUnitsData en myUnits
    private bool LoadUnits(){}

    // método publico que da acceso a los datos de cualquier unidad a partir de
    su id
    public UnitData GetUnitData(UnitId anId){}

    //...
}
```

### 2.7.2.3 Game Object Factory

**Duración estimada:** 4d

Ya con los datos que refieren a entidades de juego definidos y cargados, desarrollamos un sistema para instanciarlos en una escena. Para esto usamos el patrón de fábrica, tenemos una clase llamada `SharedGameObjectFactory` accesible por todas las clases en ejecución en cualquier escena. Esta clase se comunica con `SharedDataLoader` y tiene la responsabilidad de instanciar, inicializar y devolver *game objects* a partir de un identificador y datos contextuales como posición y relaciones.

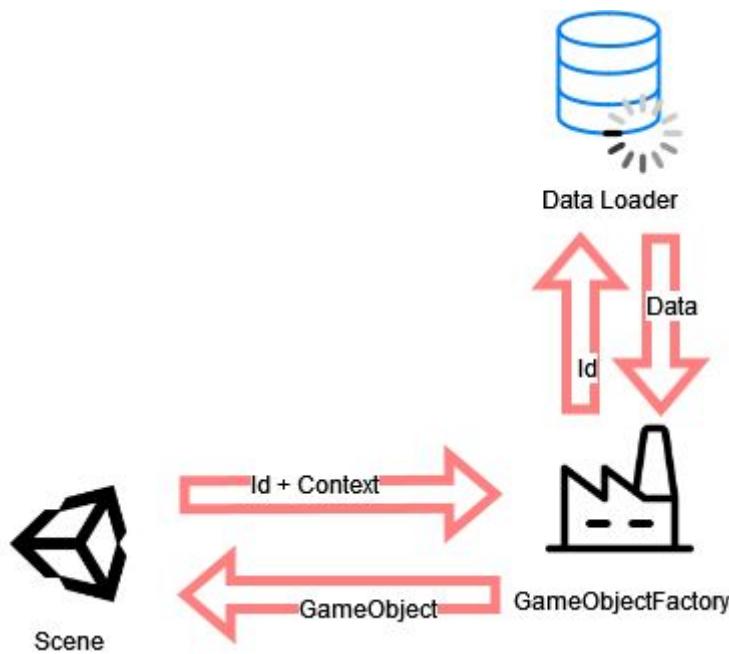


Figura 58. Game Object Factory

```

public SharedUnit CreateUnit(Transform aParent, Vector3 aPosition, SharedPlayer
anOwnerPlayer, bool aIsCapitain, Vector2Int aBoardPosition, UnitId anId, int
aMatchId)
{
    if (!IsDataLoaded() || anId == UnitId.INVALID || !myHasLoader)
    {
        // Salida temprana si no tenemos datos básicos necesaria para continuar
        Shared.LogError("[HOOD][FACTORY] - CreateUnit");
        return null;
    }

    // Llamada a método de Unity Instantiate, recibe el objeto prefab, una
    // posición, una rotación, y una relación de dependencia en el árbol de jerarquía
    SharedUnit unit = Instantiate(myUnitPrefab, aPosition, Quaternion.identity,
aParent);

    // Buscamos los datos específicos para este identificador
    UnitData data = myDataLoaderReference.GetUnitData(anId);
    if (data != null)
    {
        // Método a medida para inicializar este GameObject, podemos pasar
        // relaciones y los datos de esta unidad
        unit.Init(anOwnerPlayer, aIsCapitain, aBoardPosition, data, aMatchId);
        // En este punto la unidad esta instanciada y en escena.
        return unit;
    }

    Shared.LogError("[HOOD][FACTORY] - CreateUnit, no data from DataLoader");
}

```

```
    return null;  
}
```

## 2.7.3 Desarrollo del Gameplay

**Duración estimada:** 14d

Al momento de desarrollar las funciones necesarias para cumplir con nuestras expectativas con respecto al *gameplay*, fue necesario determinar qué acciones podría realizar el usuario y desde ese punto determinar de qué manera se lograría hacer en código y en el editor de *Unity*. El qué fue explicado anteriormente, por lo que esta sección se enfocará en las soluciones encontradas para llevar estas acciones a la realidad.

### 2.7.3.1 Objetos de Gameplay

A continuación se describirán en detalle los objetos que existen en la etapa de juego que permiten y ayudan a los usuarios a llevar a cabo estas acciones:

#### Carta

El *prefab* “Card” fue creado, y en éste se muestra toda la información de una carta para el usuario de forma visual. Este *prefab* tiene el *script* `SharedCard` como componente, que contiene todas las propiedades de la carta, como su nombre, costo en energía, ataque, etc. y la lógica necesaria para mostrar estos datos en el objeto dentro del juego.

Para la partida, se creó el *prefab* “MatchCard”, que hereda de “Card” y es idéntico en apariencia. De la misma manera, tiene como componente a un *script* homónimo que contiene la lógica necesaria para manipular la carta durante una partida. Este *script* asimismo hereda de `SharedCard` por lo que también permite mostrar los datos de la carta al usuario.



Figura 59. Card/MatchCard

Para manipular la carta, se utilizaron cuatro interfaces incluidas en *Unity*:

`IPointerDownHandler`, `IBeginDragHandler`, `IDragHandler` y `IEndDragHandler`.

Estas interfaces permiten detectar cuando un jugador clickea sobre la carta, cuando un jugador comienza a arrastrar a la carta mediante clickeando y moviendo el mouse sin soltar el click, arrastrar la carta en sí, y detectar cuando se terminó de arrastrar la carta respectivamente.

Todos estos momentos son importantes a la hora de determinar que tiene que hacer la carta y en qué momento.

Al comenzar a ser arrastrada, la carta debe volverse ligeramente transparente para permitir que el usuario vea el tablero de la mejor manera posible, se reseteará todo tipo de selecciones y acciones que el usuario haya hecho y se colorean las celdas donde el usuario podrá colocar la carta. Estos puntos se explicarán en más detalle más adelante.

Al terminar de ser arrastrada la carta simplemente reseteará su estado visual, dejando de ser transparente y volviendo a su posición inicial.

## Celda

Las celdas son componentes claves del juego por lo que se dedicó especial atención a su representación visual y comportamiento.

En cuanto a sus propiedades —determinadas por su tipo— las celdas tienen un costo, que permite calcular qué tanto se podrá mover una unidad en el tablero. Por ejemplo, una unidad con rango de movimiento 3, podrá desplazarse por tres celdas de costo 1, o dos celdas de costo 2 y 1. También tienen su posición en forma de vector de dos dimensiones, que permite ubicar a cada celda en particular, y ubicar a sus celdas adyacentes. A su vez tienen una propiedad que determina si esa celda permite invocar cartas, y por último si está bloqueada (algo que restringe el movimiento por completo).

Estas últimas dos propiedades no sólo se ven afectadas por el tipo de la celda, sino también por si hay una unidad ubicada en ésta.

En el juego existe la celda vacía, que sólo tiene un costo de 1, la celda nébula, que tiene un costo de 2, y la celda agujero negro, que bloquea el movimiento y la invocación por completo.

En cuanto a lo visual, las celdas se diferencian de dos maneras:

La primera es determinada por su tipo, y sirve para darle una indicación visual a los jugadores sobre qué tipo de celda están viendo.

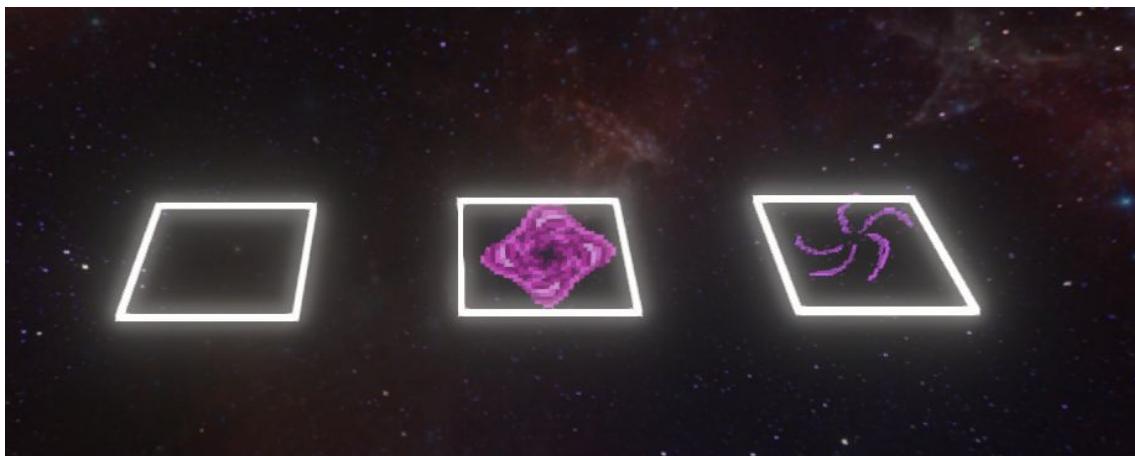


Figura 60. De izquierda a derecha: Celda vacía, celda nébula, y celda agujero negro.

La segunda diferenciación es por su color. Normalmente, todas las celdas son blancas, pero diferentes acciones pueden cambiar su color:

#### Cambio de color para un usuario

Estos cambios de color son exclusivos para cada usuario y existen para incrementar el grado de interactividad que tiene el juego y, más importantemente, para facilitar el entendimiento y la comunicación de las reglas para el usuario.

Cada vez que el usuario quiera hacer una acción que sólo es permitida en ciertas celdas, las celdas se iluminan de un color para demostrar esto. Una vez terminada o cancelada la acción, las celdas vuelven a su color original.

Para darle un sentimiento más “táctil” al juego, las celdas también cambian de color al pasar el mouse sobre ellas y al hacerles click.

#### Cambio de color para todos los usuarios

Las habilidades que cambian el comportamiento de una celda en el juego también hacen que cambie de color, un cambio que dura mientras dure el efecto. Estos cambios son vistos por ambos usuarios y si un usuario hace alguna acción como las descriptas en

la sección anterior, la celda cambia de color, pero al cancelar o terminar la acción la celda no vuelve a su color original, sino al color resultante de la habilidad.

Por ejemplo, en un turno se activó la habilidad “Protector”, que cambia el color de las celdas donde está activada del blanco al azul. Si un jugador quiere consultar a qué celdas se puede mover una unidad, las celdas válidas se colorean de amarillo, pero al terminar de moverse o si cancela la acción las celdas afectadas por “Protector” vuelven al color azul, no al blanco.

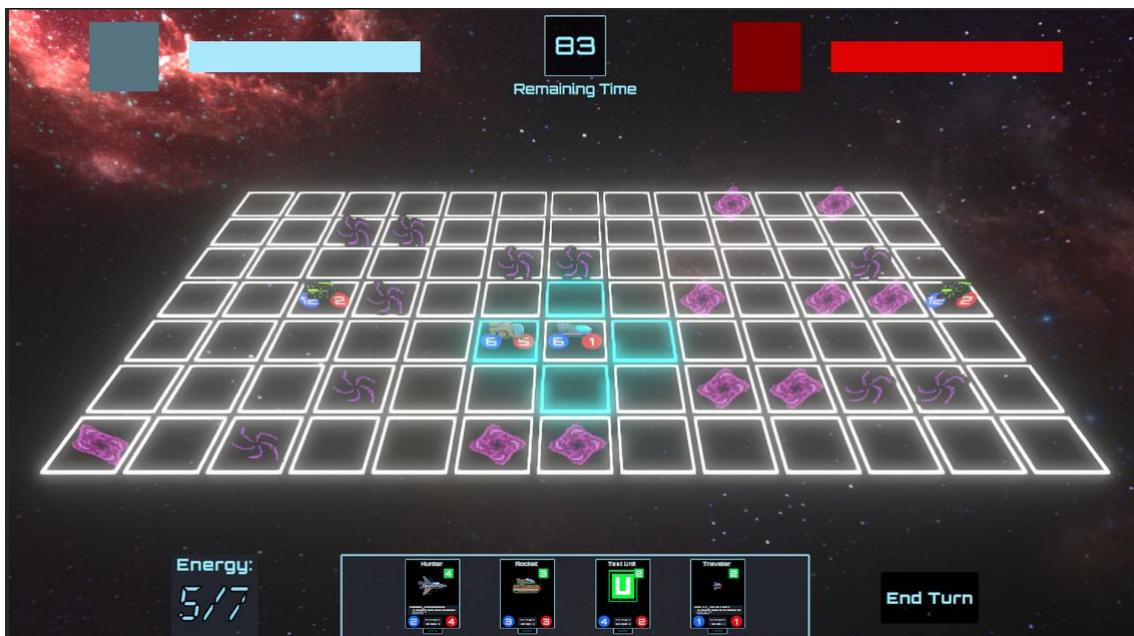


Figura 61. 1: Se activa una habilidad cambiando el color de las celdas a azul para todos los jugadores.

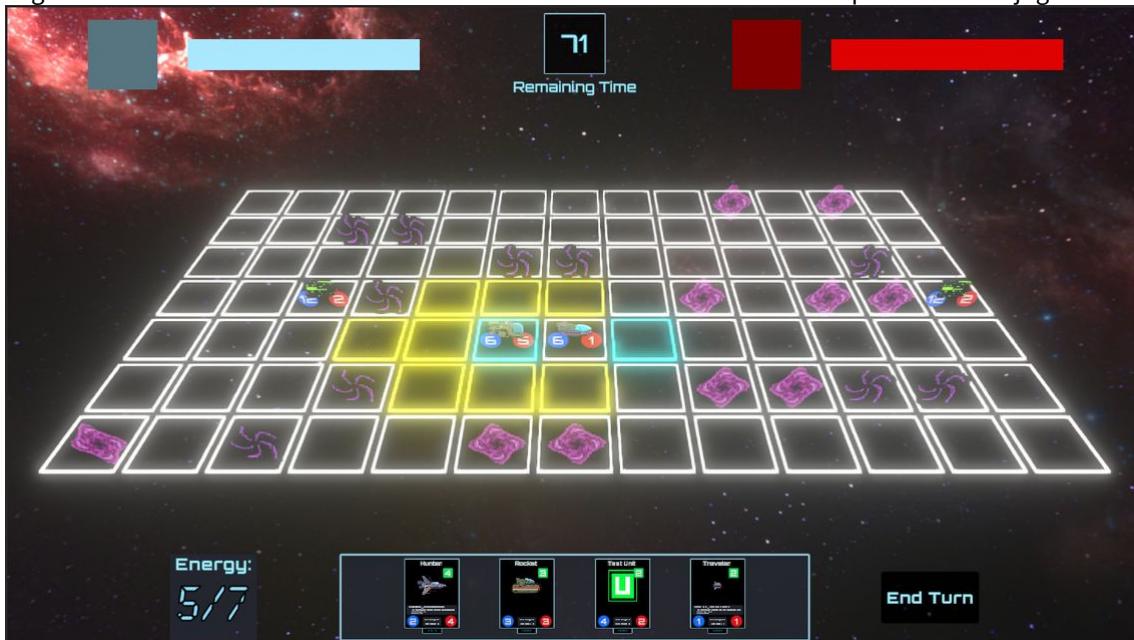


Figura 62. 2: El jugador hace click en otra unidad, mostrando los lugares a donde puede ir.

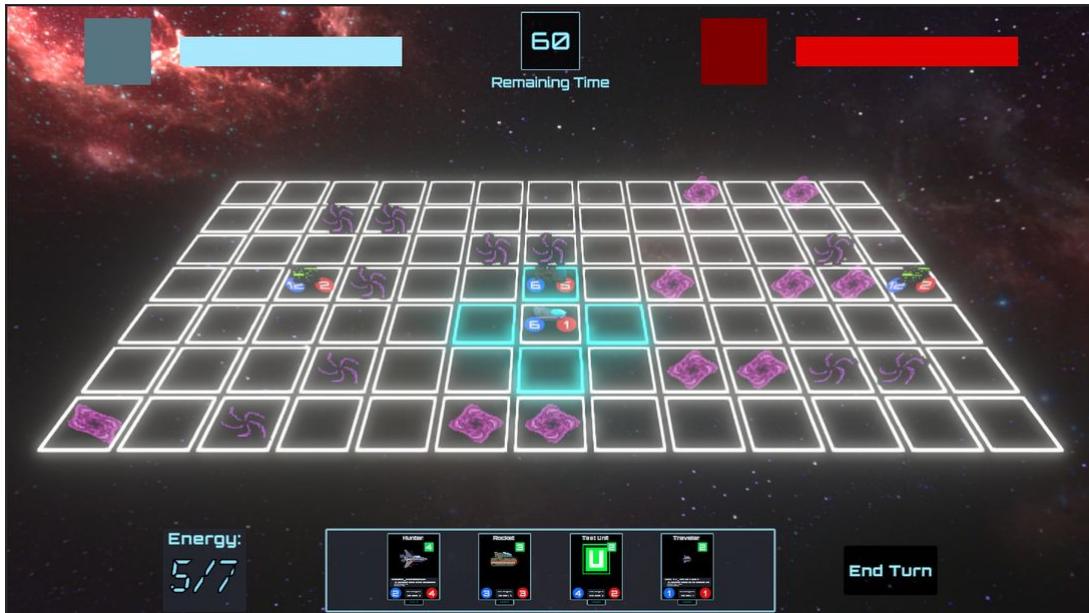


Figura 63. 3. Al mover la unidad, el tablero vuelve a su color blanco original, excepto las celdas afectadas por la habilidad, que vuelven al color azul.

Para alcanzar estos objetivos se crearon dos propiedades que manejan los colores:

`BaseColor` y `FallBackColor`. La primera determina el color base de la celda, el color que tendrá si el jugador local no está haciendo ninguna acción que cambie el color para ese usuario. La segunda se usa para preservar un color al que la celda deberá volver al terminar una acción.

Por ejemplo al pasarle el mouse por encima a una celda, está cambia su `BaseColor` a azul, pero mantiene su `FallBackColor` en blanco (o si está siendo afectado por una habilidad, el color de esa habilidad), para cuando el mouse deje pasarle por encima, la celda usa su `FallBackColor` para volver a su color anterior.

Para ejecutar esta lógica, las celdas tienen el script `SharedTile`, que contiene toda la lógica y las propiedades que les competen a las celdas. Este script también implementa cuatro interfaces de Unity: `IDropHandler`, `IPointerClickHandler`, `IPointerEnterHandler` y `IPointerExitHandler`.

- `IDropHandler` detecta cuando de suelta un objeto arrastrado encima de una celda. En nuestro caso, se usa para detectar cuando se coloca una carta en una celda. Al colocarse una carta en la celda, el *GameManager* hace lógica necesaria para determinar si esa carta puede invocarse, o si no. En caso positivo, la carta se destruye, en caso negativo, vuelve a la mano.

- *IPointerClickHandler* detecta cuando un usuario hace click en una celda. En este momento se consulta si esa celda contiene una unidad, y de ser así se calculan y determinan las acciones posibles de esa unidad —algo que se explicará más abajo. Si no hay una unidad, la celda simplemente se ilumina de color azul por un breve momento. Este método también tiene en cuenta si el usuario está tratando de usar una habilidad —véase la sección de “Habilidad”. Si es así, el juego tratará de usar la habilidad seleccionada.
- *IPointerEnterHandler* detecta cuando el mouse pasa por encima de una celda. La celda se ilumina de color azul mientras el mouse permanezca encima de la celda. Si se está ejecutando una habilidad, el tablero se iluminará con el área de influencia de esa habilidad, centrado en la celda en donde se encuentra el mouse.
- *IPointerExitHandler* inversamente detecta cuando un mouse deja de estar encima de una celda. La celda vuelve a su color base en este momento. Si se está ejecutando una habilidad, el tablero en su totalidad volverá a sus colores base cada vez que se llama a este método.

### 2.7.3.2 Acciones

#### Movimiento

Todas las unidades tienen un rango de movimiento dentro del que se pueden mover una vez por turno, mientras no hayan atacado o activado su habilidad en ese turno. Como se traduce este rango al tablero se ve afectado por dos factores: si una celda está bloqueada y el costo de moverse por ella, como fue explicado en la sección “Celda”.

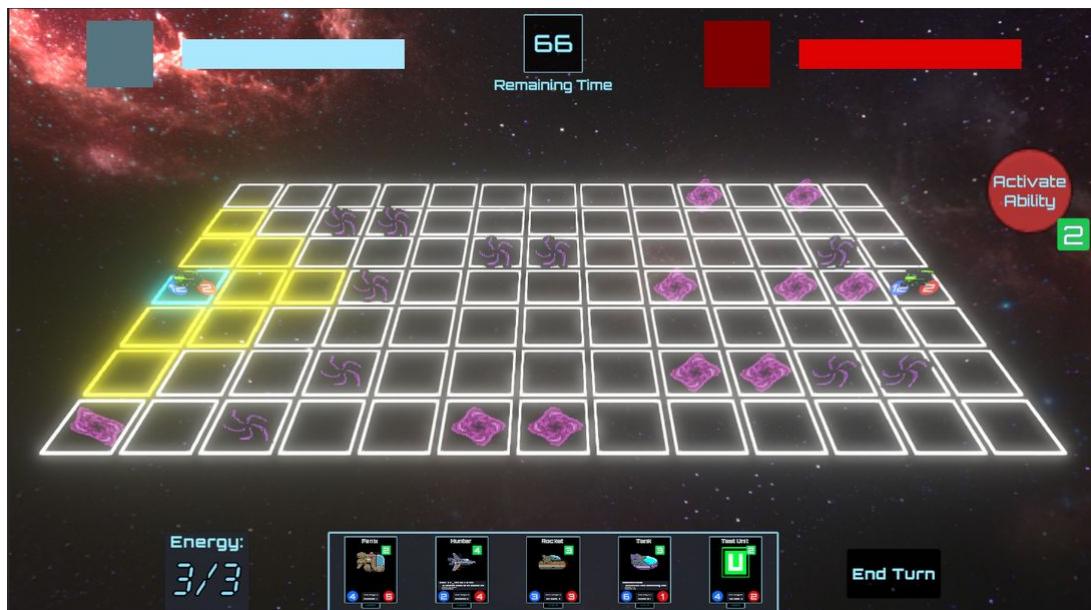


Figura 64. Rango de movimiento '2' sin obstáculos

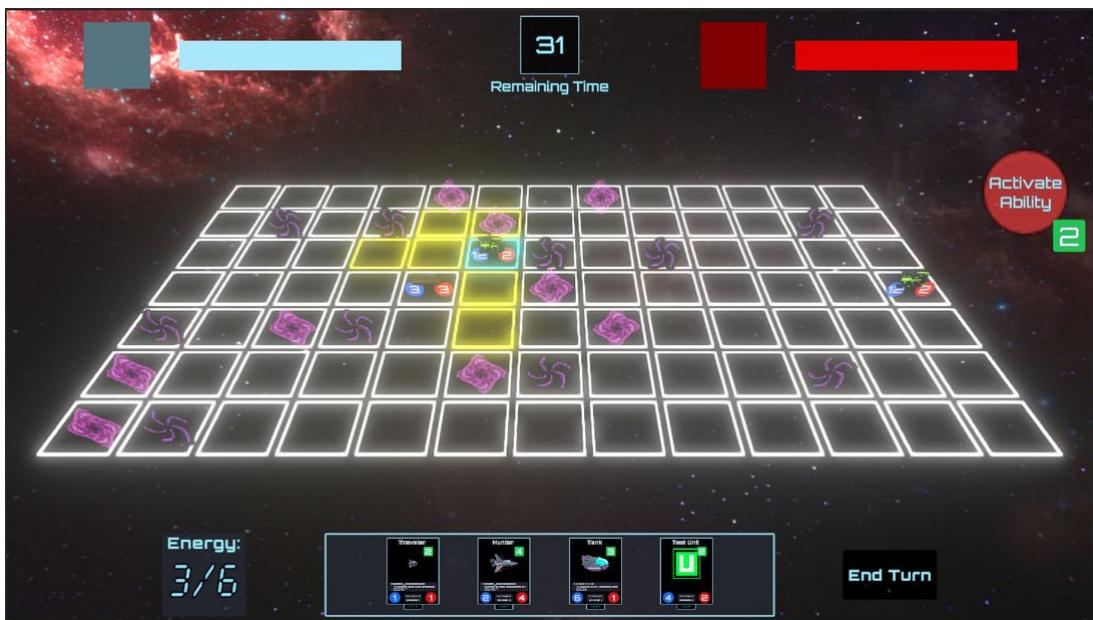


Figura 65. Rango de movimiento '2' con obstáculos

El código para el cálculo de estas celdas es el siguiente:

```
public virtual List<SharedTile> GetValidMovementRanges(SharedUnit unit)
{
    SharedUnit.MovementInfo movement = unit.GetMovementInfo();
    HashSet<SharedTile> possibleMovementTiles = new();
    Dictionary<Vector2Int, int> exploredAdjacentsWithCostDictionary = new
        Dictionary<Vector2Int, int>(); // Vector2Int => SharedTile | int => remaining
    moves when last explored
    int movementRange = movement.range;
    SharedTile unitTile = myBoardReference.GetTile(movement.pos);
    return GetValidMovementRangesRecursive(unitTile, unitTile, movementRange,
    possibleMovementTiles, exploredAdjacentsWithCostDictionary).ToList();
}
```

```
private HashSet<SharedTile> GetValidMovementRangesRecursive(SharedTile
currentTile, SharedTile previousTile, int remainingMoves, HashSet<SharedTile>
possibleMovementTiles, Dictionary<Vector2Int, int>
exploredAdjacentsWithCostDictionary)
{
    SharedTile.PathingInfo currentTileInfo = currentTile.GetPathingInfo();
    List<SharedTile> adjacentTiles =
    myBoardReference.GetAdjacentTiles(currentTileInfo.pos);
    adjacentTiles.Remove(previousTile);

    foreach (SharedTile adjacentTile in adjacentTiles)
```

```

{
    SharedTile.PathingInfo adjacentTileInfo =
adjacentTile.GetPathingInfo();

    if (!adjacentTileInfo.isBlocked)
    {
        int updatedRemainingMoves = remainingMoves - adjacentTileInfo.cost;

        if (updatedRemainingMoves > 0) // If I still have available moves
        {
            if
(exploredAdjacentsWithCostDictionary.TryGetValue(adjacentTileInfo.pos, out int
previousRemainingMoves))
            {
                if (previousRemainingMoves < updatedRemainingMoves)
                {

exploredAdjacentsWithCostDictionary[adjacentTileInfo.pos] =
updatedRemainingMoves; //Update the dictionary value
                }
                else // If there is a value in the dictionary, but it has
more remaining moves than this tile
                {
                    continue;
                }
            }
            possibleMovementTiles.Add(adjacentTile); // Hashsets don't add
duplicates, so no need to check here.

exploredAdjacentsWithCostDictionary.TryAdd(adjacentTileInfo.pos,
updatedRemainingMoves);

            GetValidMovementRangesRecursive(adjacentTile, currentTile,
updatedRemainingMoves, possibleMovementTiles,
exploredAdjacentsWithCostDictionary);
        }
        else if (updatedRemainingMoves == 0)
        {
            possibleMovementTiles.Add(adjacentTile);
        }
    }
}
return possibleMovementTiles;
}

```

## Invocación

Cada jugador comienza el juego con una nave nodriza lista para usarse, pero el usuario tiene la posibilidad de invocar otras unidades en el tablero.

Para este fin se debió permitir al usuario tomar una carta, moverla al tablero, y al soltarla que una unidad aparezca correctamente en éste. También se debió determinar en qué celdas es posible invocar una carta, mostrarle esto al usuario y determinar un costo para esta acción.

En su turno, mientras tenga suficiente energía para pagar por la carta, un jugador puede invocar en celdas no bloqueadas que rodean a su nave nodriza, o en las celdas adyacentes de unidades con el atributo `CanSpawn = true`.

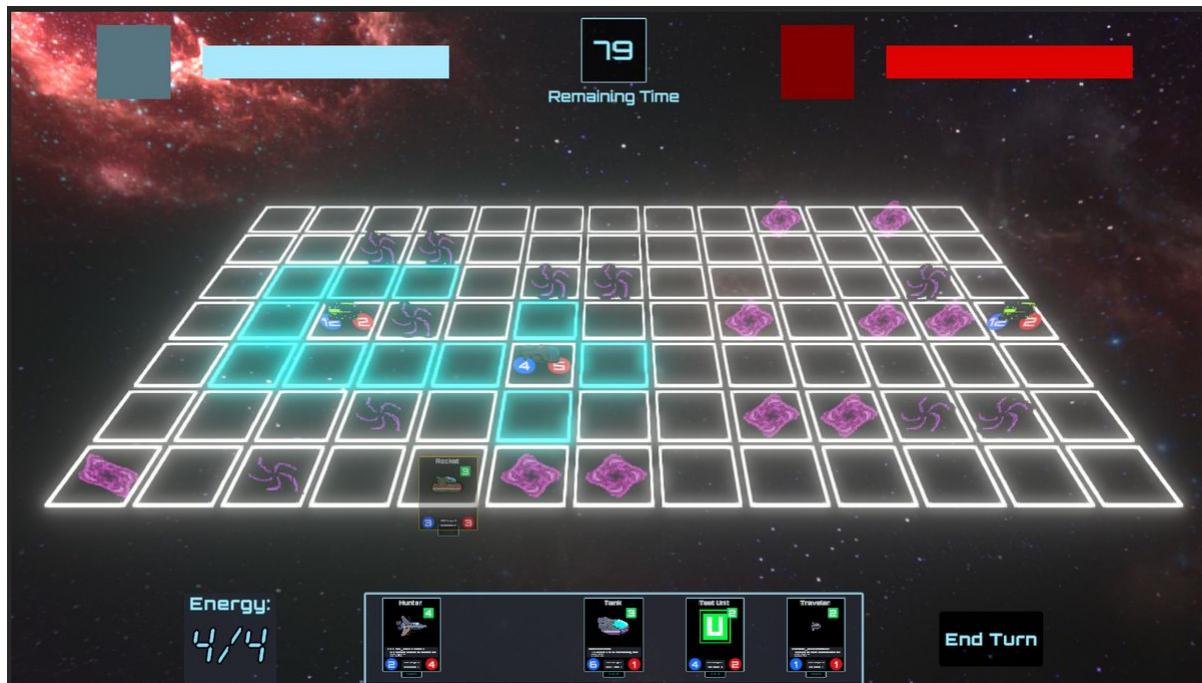


Figura 66. Las celdas válidas para invocar una nueva carta se marcan en azul.

## Ataque

Toda unidad puede atacar a otra unidad enemiga una vez por turno, siempre y cuando no haya activado su actividad en ese turno. Al atacar, ambas unidades pierden en vida (*shield*) lo que la unidad contraria tiene en ataque. Al final de esta acción, toda unidad que tenga 0 o menos de vida, muere, y deja de existir en el tablero.

Para determinar dónde puede atacar cada unidad tiene un rango de ataque que se aplica en cuatro direcciones: arriba, abajo, derecha e izquierda. Si una unidad enemiga se encuentra en este rango, puede ser atacada, pero si existe cualquier entidad entre el atacante y el atacado que bloquee una celda, como otra unidad o un agujero negro, el ataque no podrá ser efectuado en cualquier celda por detrás de la entidad.

Las celdas nébula también afectan el rango de ataque. Mientras bloquean ataques por completo, sí restan 1 al rango total de ataque en la dirección de estas celdas.

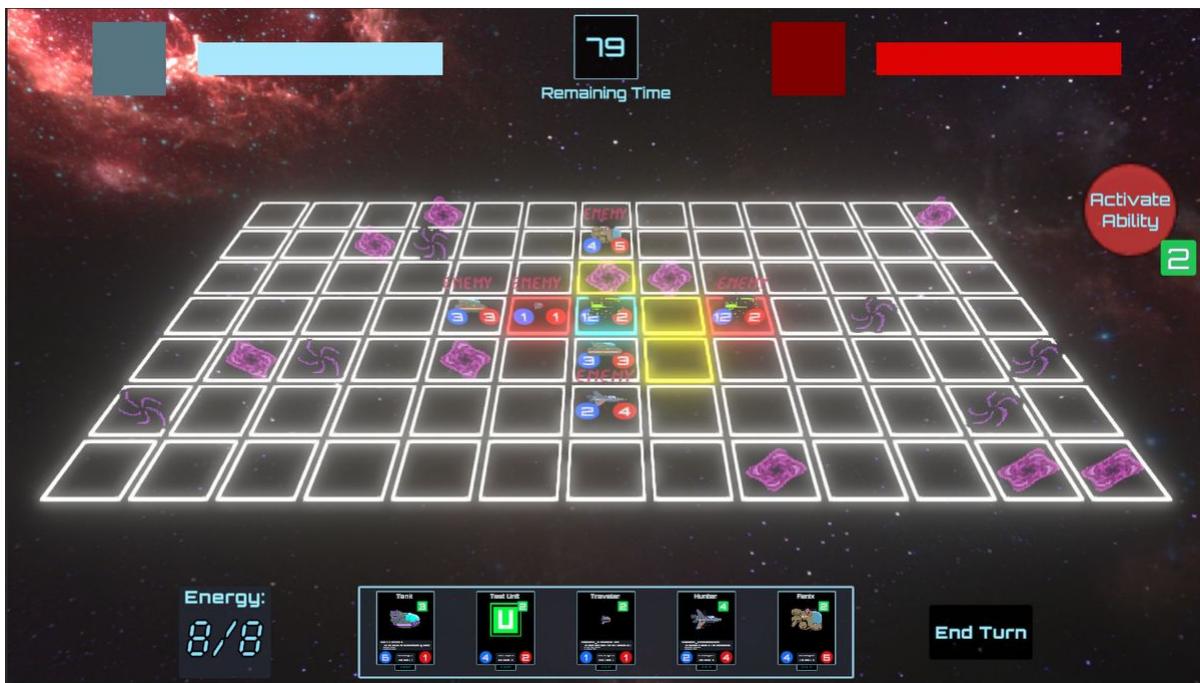


Figura 67. Ejemplo: Posibles ataques para unidad de rango de ataque 2 (Marcados en rojo)

El código para determinar las celdas válidas para atacar es el siguiente:

```

public virtual List<SharedTile> GetValidAttackTiles(SharedUnit
anAttackingUnit) // This one returns just the tiles where an attack is possible
{
    if (!MyHasLoaded())
    {
        Shared.LogError("[HOOD] [GAMEMANAGER] - GetPossibleAttackTiles()");
        return null;
    }

    int attackingUnitRange = anAttackingUnit.GetAttackRange();
    List<SharedTile> possibleAttackTiles = new();

    List<List<SharedTile>> attackingTilesListsByDirection = new();
    Vector2Int[] directionHelpers = { new Vector2Int(1, 0), new
Vector2Int(-1, 0), new Vector2Int(0, 1), new Vector2Int(0, -1) };

    for (int i = 0; i < 4; i++)
    {
        List<SharedTile> rawRangeTiles =
myBoardReference.GetShapeFromCenterTileCoord(false, AreaShape.LINE,
attackingUnitRange, anAttackingUnit.GetPosition(),
anAttackingUnit.GetPosition() - directionHelpers[i]);
        attackingTilesListsByDirection.Add(rawRangeTiles);
    }
}

```

```

    }

    foreach (List<SharedTile> tileList in attackingTilesListsByDirection)
    {
        int tilesToSearchCount = tileList.Count;

        for (int i = 0; i < tilesToSearchCount; i++)
        {
            if (tileList[i].GetUnit() != null && !
tileList[i].GetUnit().IsOwnedByPlayer(myCurrentPlayerReference)) // Is this an
enemy unit?
            {
                possibleAttackTiles.Add(tileList[i]);
                break;
            }

            if (tileList[i].GetIsBlocked())
            {
                break; //You can't attack behind a blocking entity, so we
break out and continue with the next direction.
            }

            if (tileList[i].GetTileType() == TileType.NEBULA)
            {
                tilesToSearchCount--; // Range gets shortened by one aTile
            }
        }
    }

    return possibleAttackTiles;
}

```

## Habilidades

Las habilidades son el tipo de acción menos definido del juego. Todas las habilidades tienen tipos de restricciones en común, pero qué hace cada habilidad depende de cada una en particular. Todas las habilidades heredan de la clase abstracta `SharedAbility`. En esta clase existen, entre otras cosas:

- El costo de la habilidad
- La forma que tomará la habilidad en el tablero — cuadrado de tamaño 2, por ejemplo
- El color de la habilidad — como se explicó anteriormente
- El rango de la habilidad — que tan lejos una unidad puede usar esta habilidad de sí misma
- Su *cooldown* — cuantos turnos debe esperar una unidad para usar la acción nuevamente
- Su duración — cuantos turnos dura la habilidad en el tablero una vez usada

Al comenzar un turno, se recorren las unidades con habilidades activadas y agrega 1 a su *cooldown* y duración. Si la duración excede el tope determinado, la habilidad desaparece.

Dentro de la clase padre existen métodos que tienen el rol de controlar y actualizar estas propiedades, pero a su vez existen tres métodos abstractos y uno virtual que sólo tiene un chequeo inicial. Estos métodos, entonces varían según la habilidad que los implementa. Estos métodos son:

`ApplyAbilityEffect` - Este método aplica cualquier sea el efecto de la habilidad. Este método se llama al activar por primera vez a la habilidad, y al comenzar cada turno.

`ApplyVisualEffects` - Este método determina el efecto visual que sucede con la habilidad, como colorear las celdas en donde está activa. La llamada de este método la determina cada habilidad en particular.

`RemoveVisualEffects` - Este método remueve el efecto visual. Se llama cuando la habilidad agota su duración.

`UpdateAbilityStatus` - Este método existe en caso de que la habilidad necesite hacer algo después de una acción del juego, como el movimiento de una unidad.

#### Habilidad ejemplo 1: Protector

*Protector* agrega 2 de vida a toda unidad aliada que se encuentre en su área de influencia. Esta habilidad se mueve con la unidad que la activó.

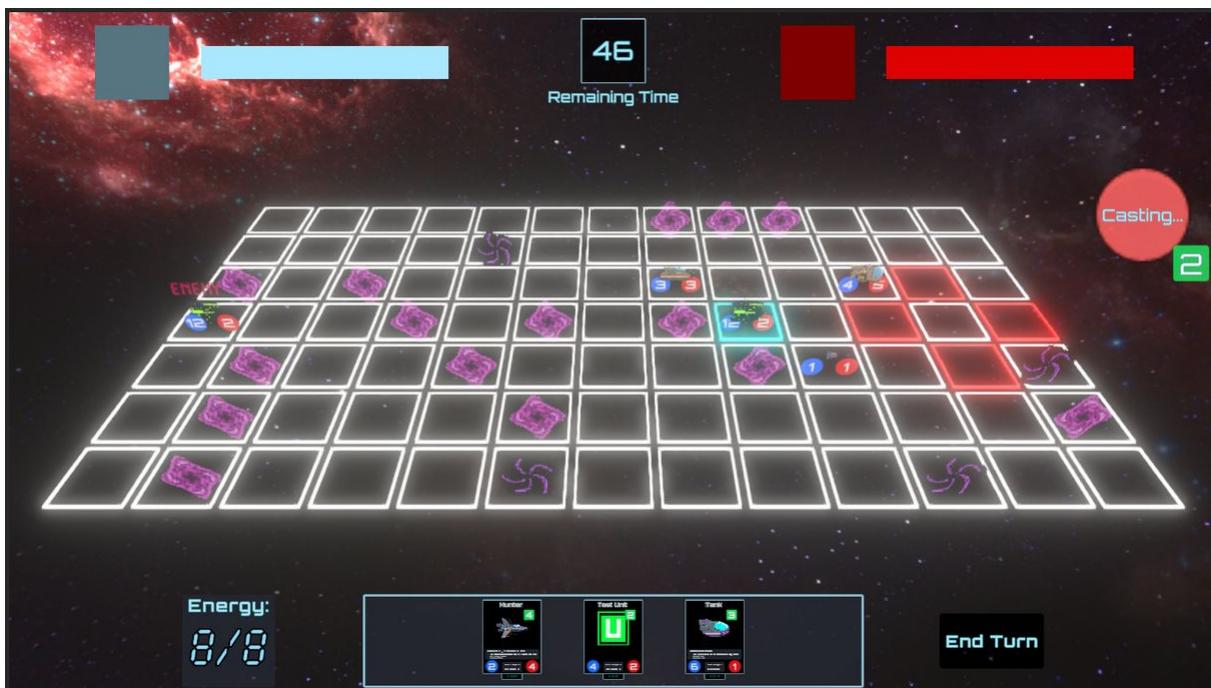


Figura 68. 1) Al comenzar a usar una habilidad, el área permitida se ilumina del color establecido por la habilidad. Al mover el mouse por el tablero, si éste se encuentra en una celda no permitida, se ilumina el tablero con la forma de la habilidad de rojo.

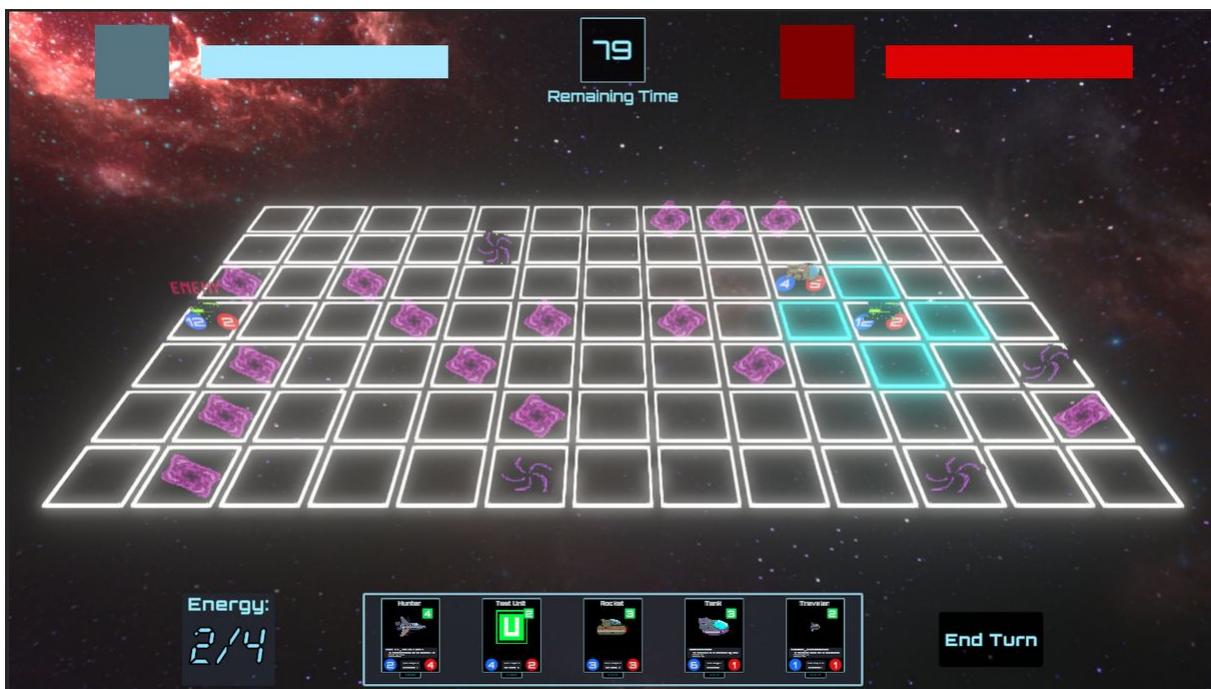


Figura 69. 2) Al posicionarse sobre la celda correcta, el color del área marcada pasa al color de la habilidad. El jugador hace click, y se activa. El jugador tenía 4 energía, y la habilidad costó 2.

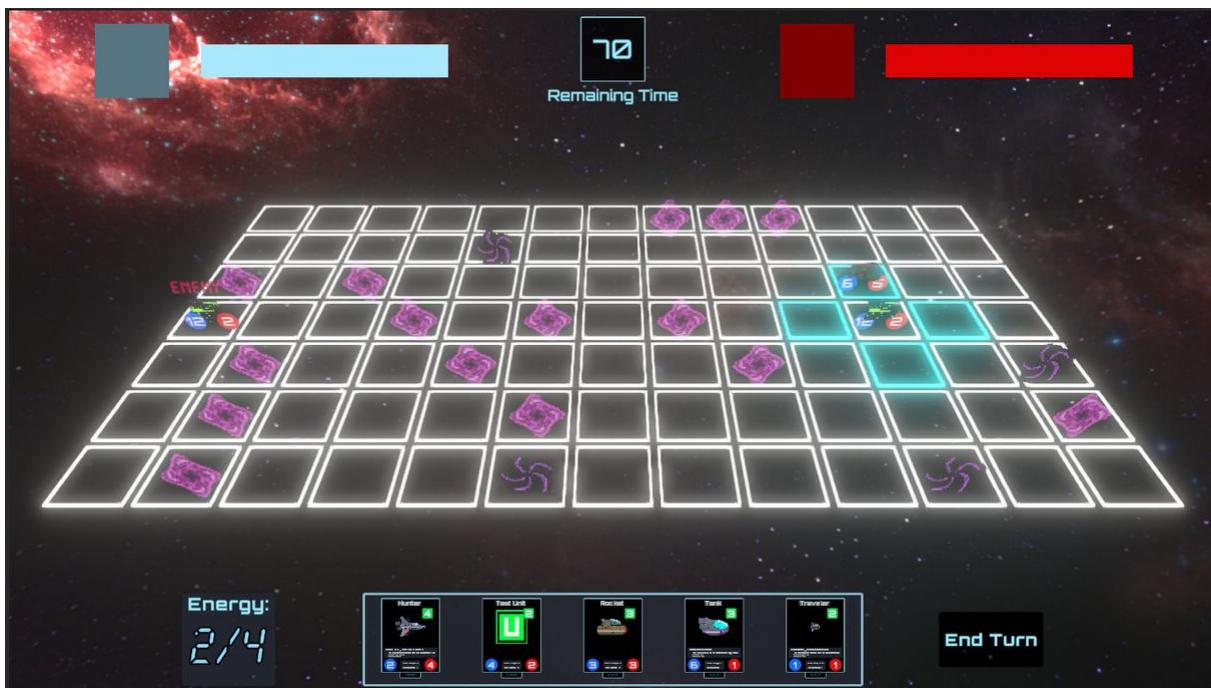


Figura 70. 3) En el mismo turno una unidad aliada se mueve a una celda afectada por la habilidad.

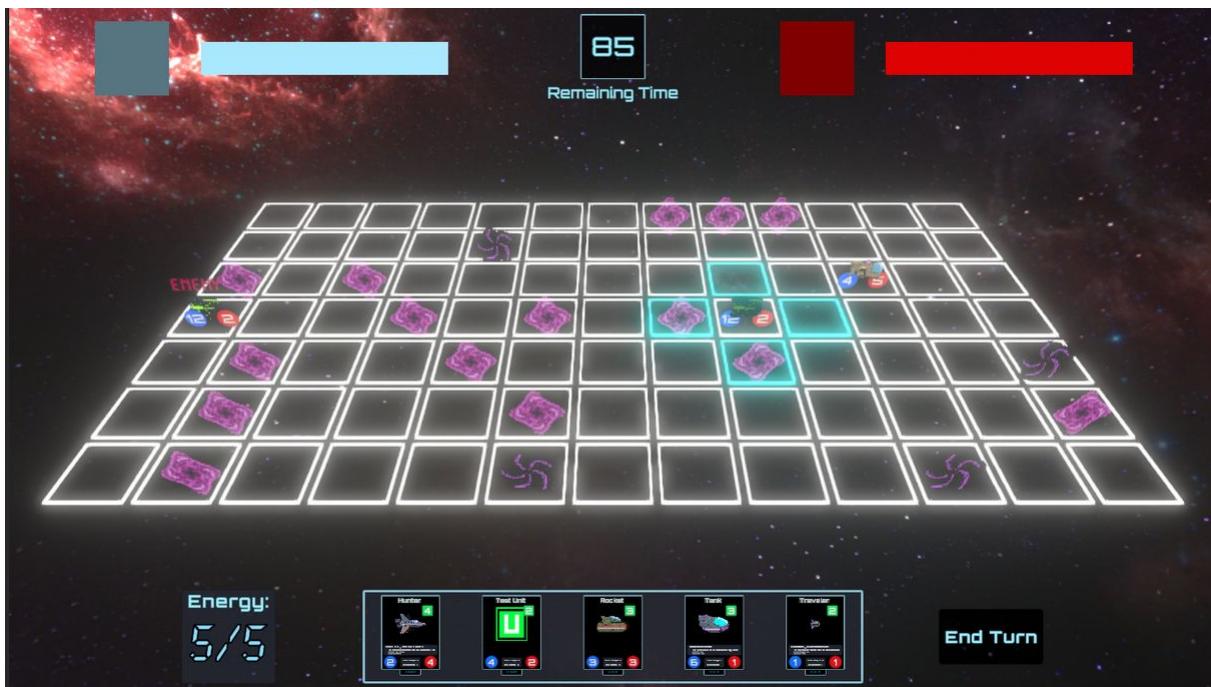


Figura 71. 4) En el siguiente turno la unidad que activó la habilidad se mueve. Protector sigue a la unidad que la activó, y entonces se mueve con ella. Al dejar de estar en el área afectada, la unidad aliada pierde su extra de vida.

## Habilidad ejemplo 2: Engine Overdrive

*Engine Overdrive* le agrega 2 de rango de movimiento a la unidad que la activó.

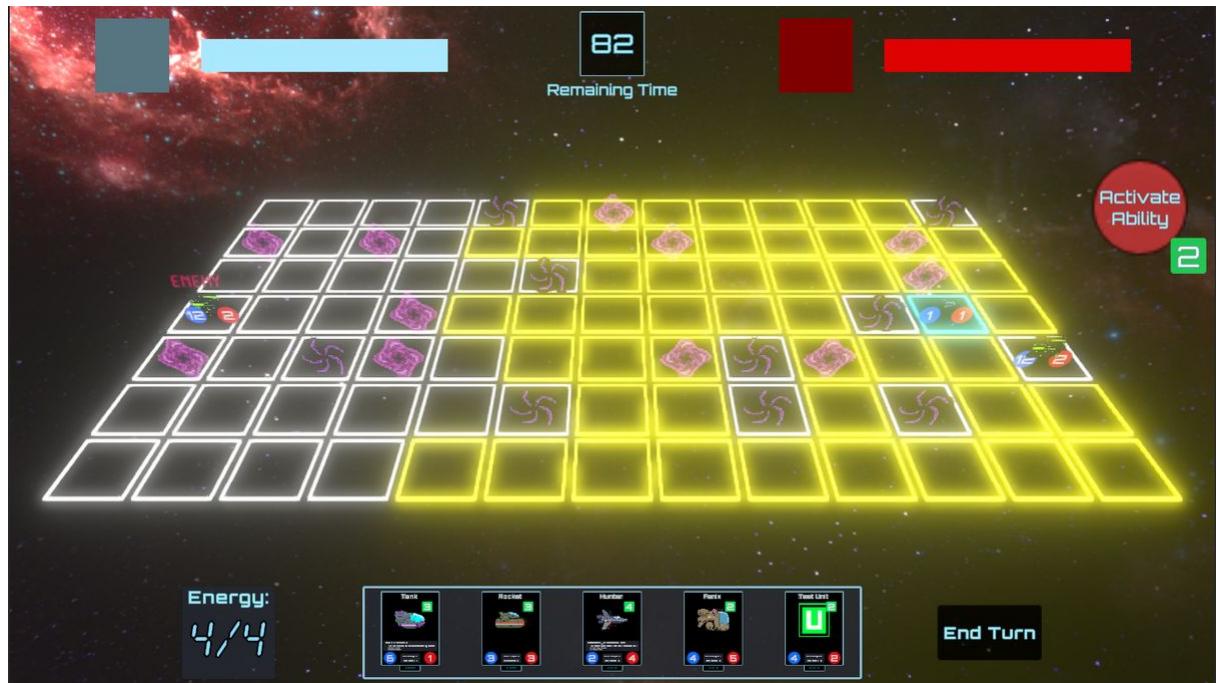


Figura 72. Rango de movimiento de la unidad antes de activar la habilidad.

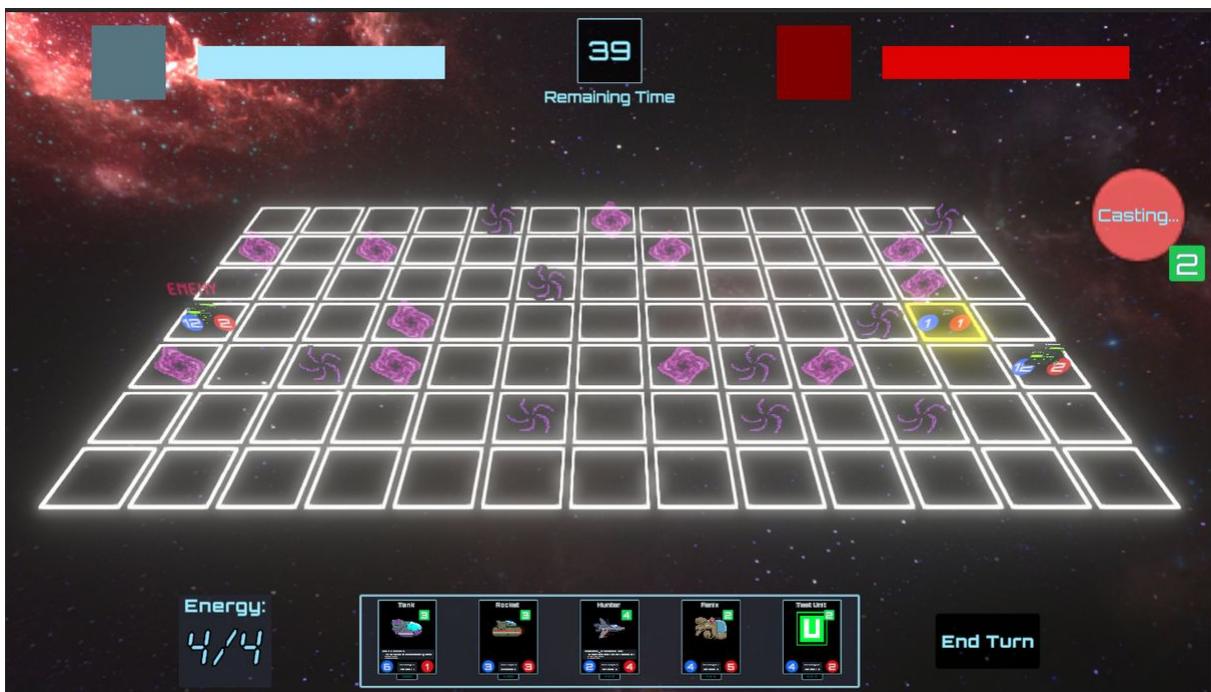


Figura 73. Activando habilidad “Engine Overdrive”

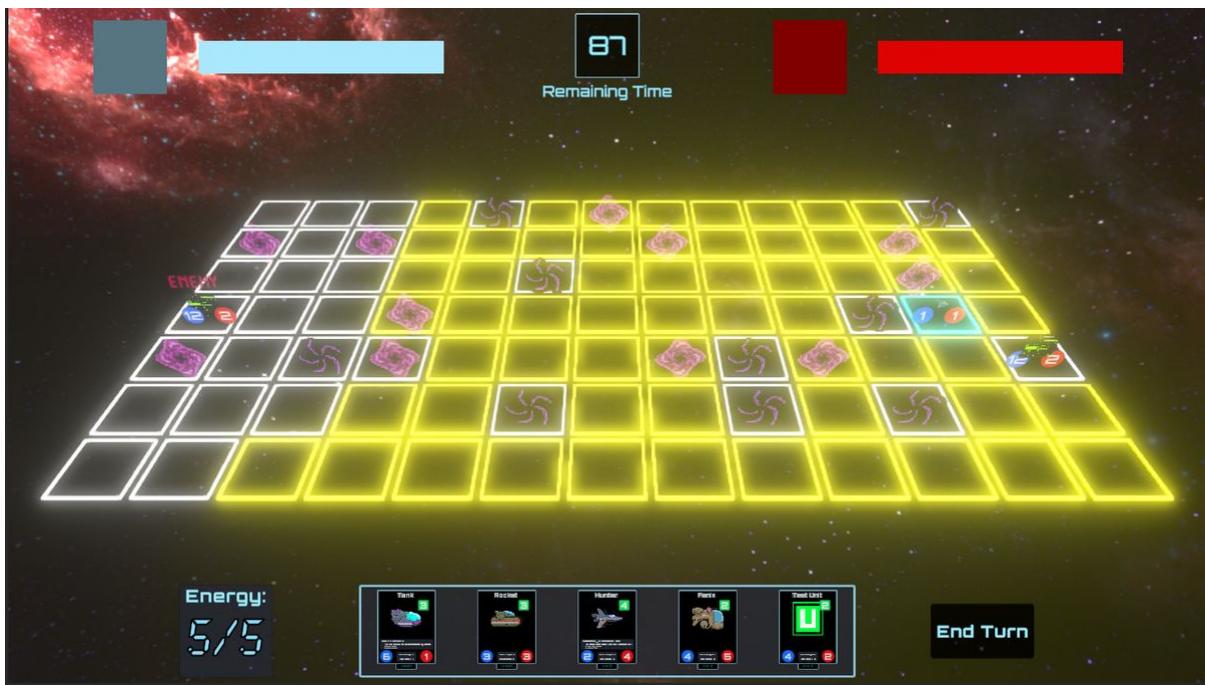


Figura 74. Rango de movimiento después de activar la habilidad.

## Status Effects

Los *status effects* son modificadores que afectan los atributos de una unidad. Una unidad puede tener muchos *status effects* distintos al mismo tiempo, pero sólo uno de cada tipo a la vez. Éstos tienen una duración que indica cuánto turnos puede pasar hasta que el efecto desaparezca. Al igual que las habilidades, se recorren las unidades con *status effects* activos y agrega 1 a su duración. El estado de esta duración se chequea cada vez que una unidad se mueve y al comenzar un turno.

Todos los *scripts* de *status effects* heredan de la clase abstracta `SharedStatusEffects`. Los métodos abstractos de esta clase siguen casi el mismo diseño que los de las habilidades y cumplen la misma función:

`ApplyEffect` - Este método aplica cualquier sea el efecto de la habilidad.

`RemoveEffect` - Este método revierte el efecto del método anterior.

`ApplyVisualEffects` - Este método determina el efecto visual que sucede con el *status effect*, en caso de ser necesario.

`RemoveVisualEffects` - Se llama cuando el *status effect* agota su duración.

La razón por la creación de esta entidad es que permite un control más preciso sobre los efectos que pueden provocar una habilidad. Por ejemplo, con `ApplyEffect` y `RemoveEffect`, es fácil revertir el efecto de una habilidad en la unidad correspondiente, ya que el *status effect* ahora se encuentra guardado en la unidad. A pesar de que simplifican el diseño y la aplicación de las habilidades, esto no significa que una habilidad necesariamente necesita aplicar un *status effect*, sin embargo.

De hecho, ambas habilidades descritas anteriormente no modifican los atributos mencionados directamente, sino que aplican un *status effect* particular que es que termina modificándolos.

En el caso de *Protector*, cualquier unidad que entre en su área de influencia recibirá el efecto *Protector Aura*. Este efecto tiene duración 0, lo que significa que al momento que una unidad no se encuentra más en el área de influencia, el efecto desaparece. Si una unidad permanece en el área al terminar el turno, el efecto se quita, pero inmediatamente se reaplica, por lo que el usuario no ve ninguna diferencia.

En *Engine Overdrive*, se aplica un efecto homónimo de duración de 2 turnos. Esto permite a que el incremento en el rango de movimiento permanezca en la unidad incluso al moverse a otra celda distinta a donde aplicó el efecto, y que se remueva al agotarse.

### 2.7.3.3 Manejo de Acciones

En las secciones anterior se describieron todas las acciones posibles en el juego y la lógica detrás de ellas. Sin embargo, un jugador debe ser capaz de llevar estas acciones a la realidad. Para este fin, se construyeron varios objetos de UI para permitirle al usuario jugar al juego de forma intuitiva y poco intrusiva.

Un manejo de acciones ya fue mencionado en la sección de “Carta” por lo que no será necesario hacerlo nuevamente. A continuación se describirán los dos botones principales disponibles para el usuario: *Action Prompt*, *Ability Prompt*, y un elemento de UI auxiliar para facilitarle el manejo al usuario: *Prompt and Selection Re-setter*.

#### Action Prompt

*Action Prompt* es un botón que permanece invisible hasta que el usuario hace click derecho sobre una celda. El juego entonces determina si esa celda pertenece a las celdas de movimiento o las celdas de ataque de la última unidad seleccionada del jugador, y si es así muestra como texto “Move” o “Attack” respectivamente. En caso de no pertenecer a estas listas se muestra “No Actions”.

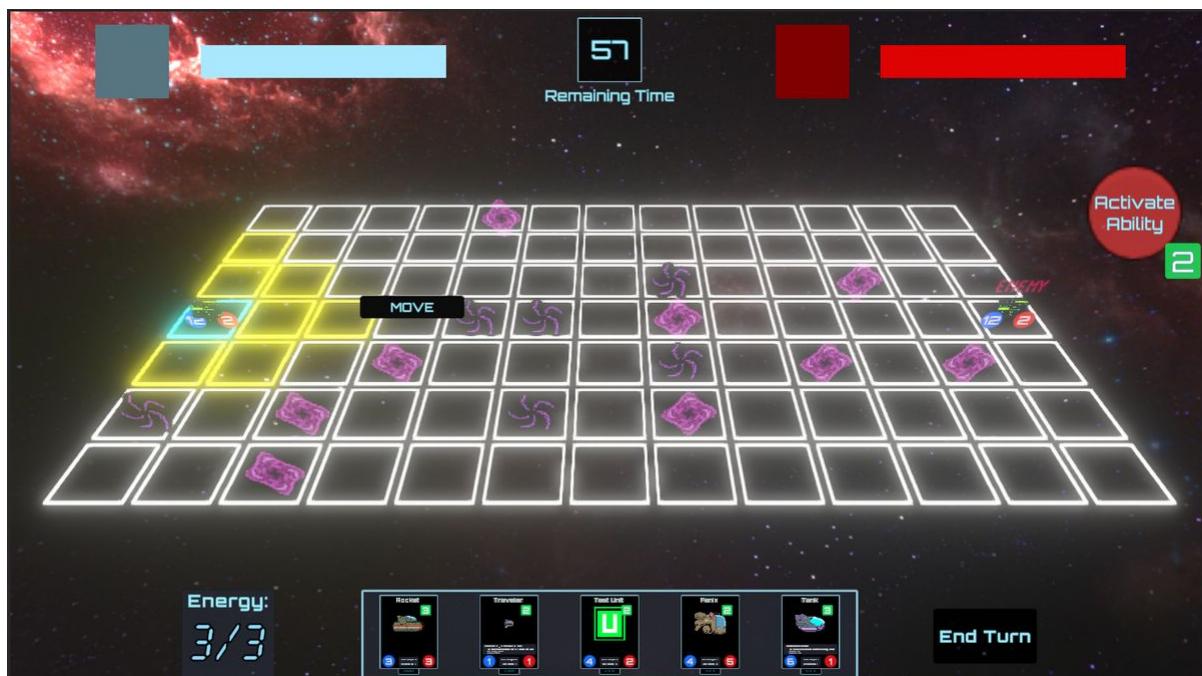


Figura 75. Action Prompt de “Move”

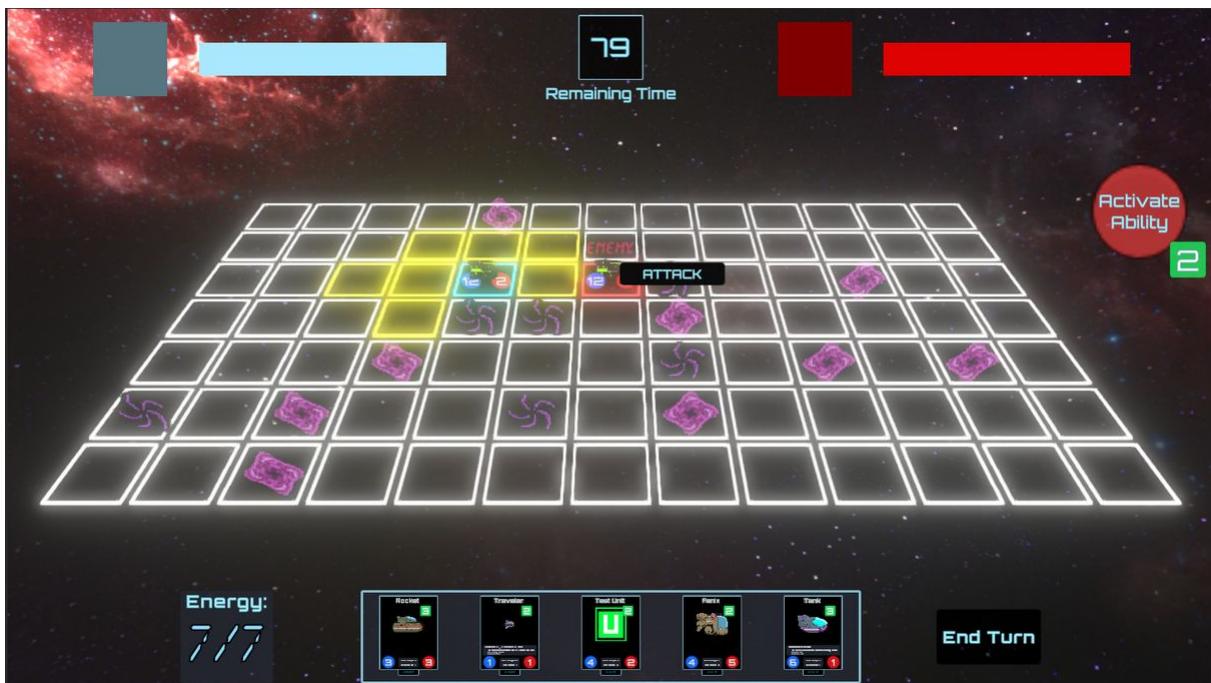


Figura 76. Action Prompt de “Attack”

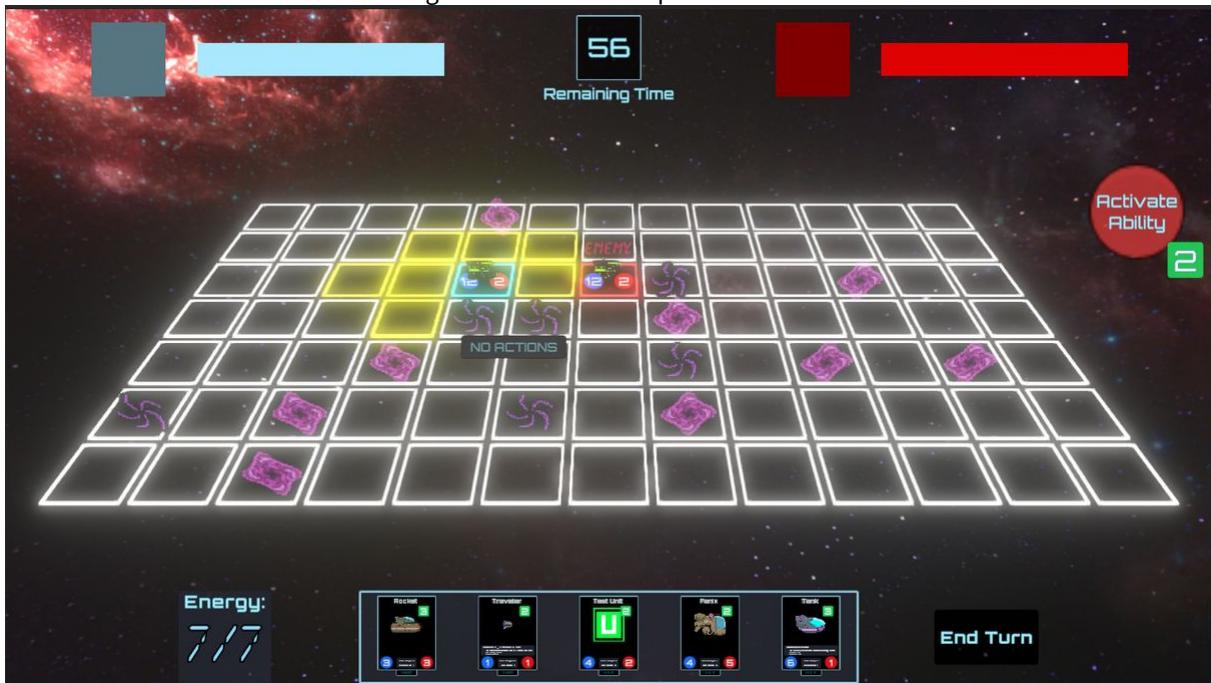


Figura 77. Action Prompt sin acción disponible

Basado en qué celda se apretó el botón, éste también cambia a un estado del mismo nombre que lo que aparece en el botón. Al apretarlo, tanto en *Move* como en *Attack*, el botón manda el *request* respectivo, basándose en su estado.

## Ability Prompt

*Action Prompt* es un botón que aparece cuando se selecciona una unidad con habilidad, que no está con su *cooldown* en progreso, y con suficiente energía para pagar esa habilidad. En la esquina inferior izquierda del botón aparece un número que indica el costo de activar esa habilidad.

Al apretarse, cambia el valor del boolean `myIsCastingUnitAbility` en `ClientGameManager` al contrario de su valor anterior. Este boolean es el que toman en cuenta las celdas para determinar si deben colorear el área de una habilidad o no al pasar el mouse por arriba.

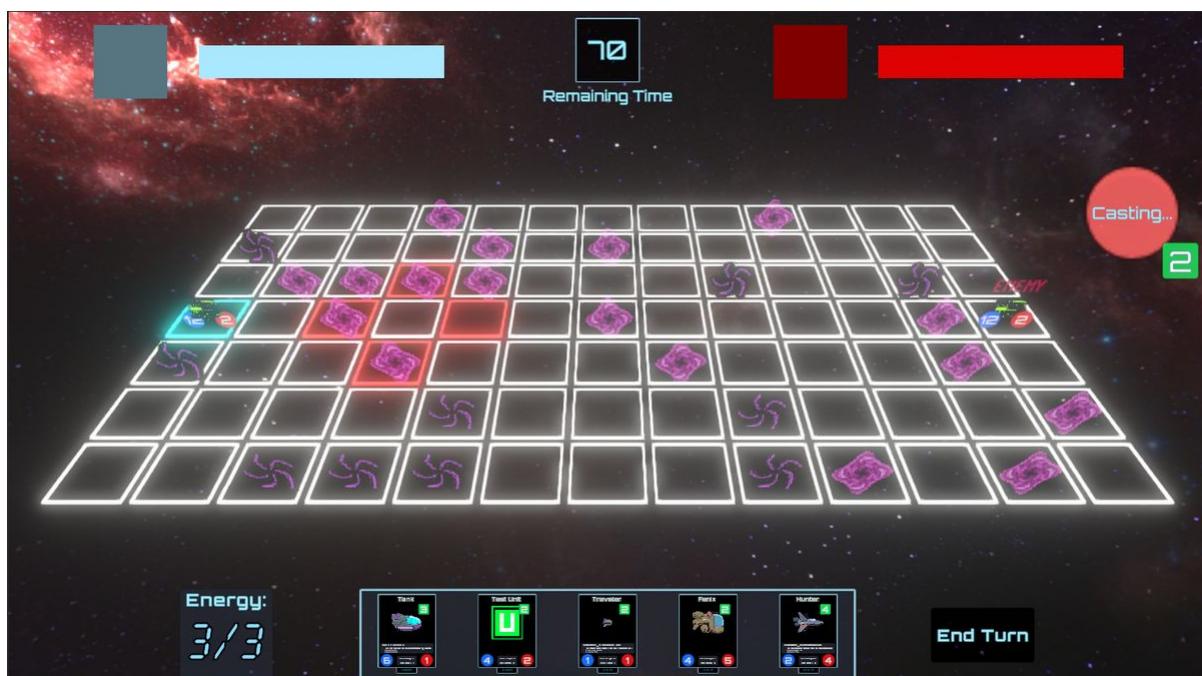


Figura 78. Como el usuario apretó el botón, éste cambia a “Casting” (esquina superior derecha). A su vez, al pasar el mouse por las celdas, se dibuja la potencial área de influencia de esta habilidad en el tablero y también se marca la celda donde esta habilidad será válida

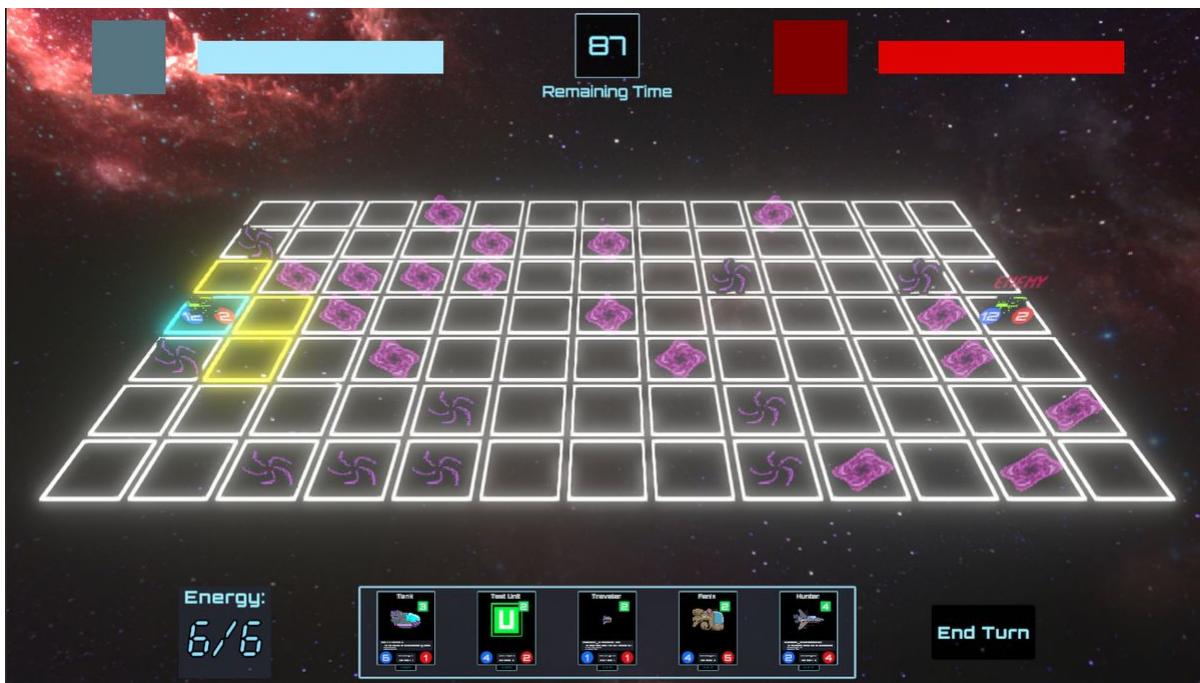


Figura 79. En esta imagen se está seleccionando a la misma unidad que usó su habilidad en la anterior, sólo que unos turnos después. Sin embargo, como está en el proceso de cooldown, el botón no aparece.

### Prompt and Selection Re-setter

La función del *Prompt and Selection Re-setter* es por sobre todo para la comodidad del usuario. Su trabajo es detectar clicks fuera del tablero o cualquier elemento de UI y resetear los colores del tablero a sus colores base y cualquier selección que haya hecho el usuario.

El reseteo de los *inputs* al hacer click en zonas “vacías” es algo que cualquier usuario de PC estaría acostumbrado y es una función extremadamente popular en juegos que utilizan el mouse para seleccionar elementos de un juego.

Por estas razones se consideró que era necesario incluir este elemento.

## 2.8 Finalización 2

### 2.8.1 Esfuerzo de Integración

**Duración estimada:** 9d

Durante la [re-estructura del ambiente de desarrollo](#) paralelizamos tareas referidas al gameplay y al diseño de la comunicación entre servidor-cliente. Sabíamos que eso implicaría un gran esfuerzo y destinación de recursos debido a que al momento de

integrar lo trabajado durante ese tiempo de *refactor* iban a surgir una suma importante de inconsistencias entre un código y el otro.

De todas formas consideramos apropiado no detener el desarrollo de las nuevas funcionalidades. Al momento de integrar el código se debieron tener en cuenta múltiples factores tales como el cambio de archivos de lugar, el cambio de propiedades (tipo y/o nombre), re-diseño de clases y métodos. Resultó en una gran cantidad de tiempo invertido, pero entendimos que era uno de esos casos donde había que dar un paso atrás para dar dos para adelante.

## 2.8.2 Comunicación Entre Clientes y Servidor

**Duración estimada:** 5d

En un juego *online* es realmente importante una comunicación fluida entre los distintos clientes y el servidor. Nuestra intención era generar un sistema de mensajes escalable que permitiera enviar información desde el cliente al servidor y viceversa.

El servidor, además de proveer información, establecer las reglas, manejar los estados del juego, es quién se encarga de validar las acciones de los distintos clientes. Es importante que sea de esta manera debido a la seguridad y prevención de trampas. De todas formas realizamos una validación inicial desde el lado del cliente. Validamos que cumpla con la lógica y reglas del juego, habilitamos o deshabilitamos botones y mostramos indicadores visuales. Si la validación inicial resulta positiva el cliente procede a dar inicio al flujo de mensajes.

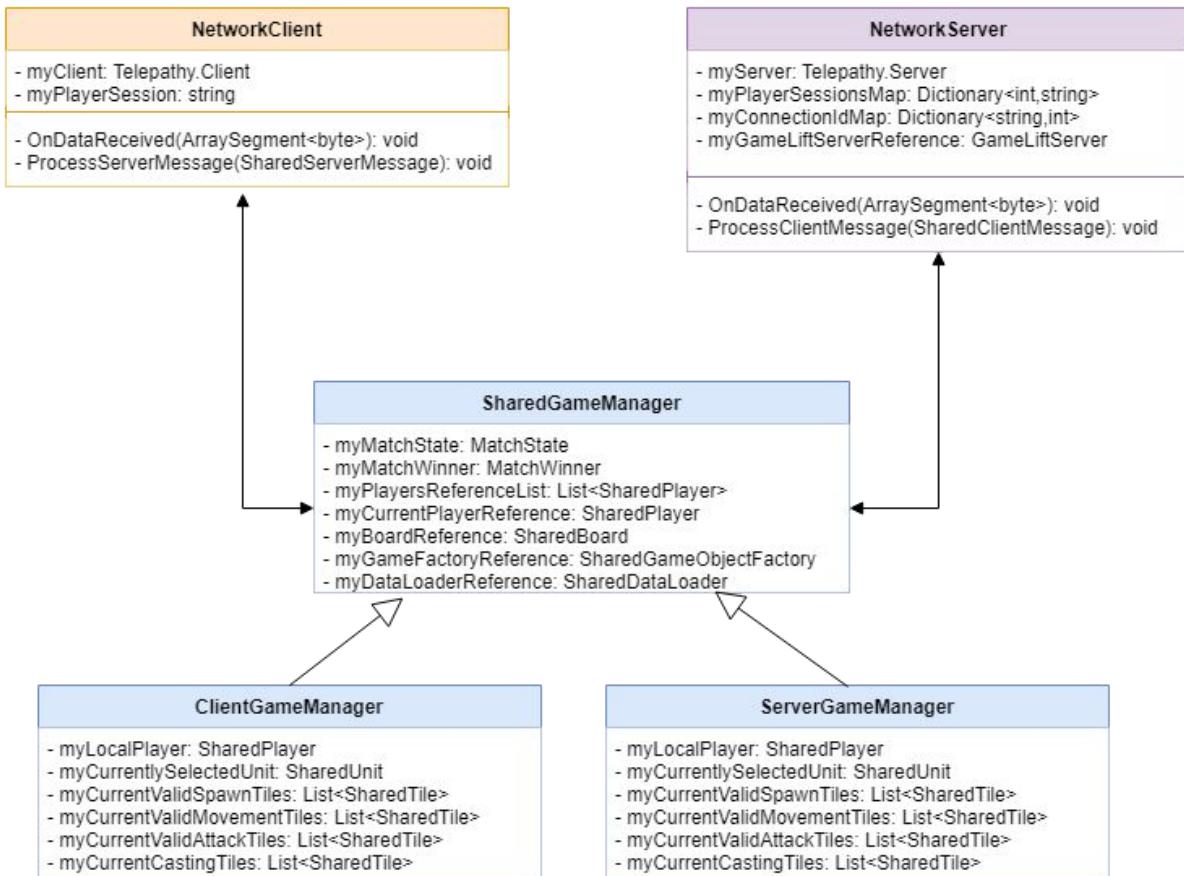


Figura 80. Diagrama de clases de comunicación entre cliente-servidor con sus propiedades principales  
 Para lograr este objetivo, creamos dos clases abstractas (`SharedClientMessage` y `SharedServerMessage`) en el código compartido, de forma tal que tanto servidor como cliente pudieran enviar o procesar estos mensajes.

`SharedServerMessage` es el tipo de mensaje base que envía el servidor, que luego será procesado por el cliente.

```

public abstract class SharedServerMessage
{
    public abstract string myType { get; }

    public SharedServerMessage()
    {
    }
}
  
```

Por otro lado, `SharedClientMessage`, es el tipo de mensaje base que envía el cliente y que luego será procesado por el servidor.

```

public abstract class SharedClientMessage
{
    public abstract string myType { get; }

    public SharedClientMessage()
    {
    }
}

```

Ambos mensajes poseen como única propiedad un *string* llamado “myType”, que será sobrescrito por las clases derivadas tomando el valor del nombre literal de dicha clase. Esto nos permite identificar el tipo específico de mensaje recibimos al momento de procesar el mensaje y poder obtener todas las propiedades de ese mensaje particular.

A su vez, implementamos los métodos que se encargan de enviar y recibir estos mensajes. En el caso de los clientes, para enviar mensajes sólo necesitan tener una operación ya que el servidor es único. Distinto es en la situación del servidor, que debe tener la posibilidad de mandarle mensaje a uno o todos los clientes dependiendo del caso. Cada una de estas operaciones consiste en serializar el mensaje que recibe por parámetro y transformarlo al formato JSON, luego codificarlo, obtener los *bytes* y enviarlos en un buffer de formato *ArraySegment*.

```

public void SendMessageToServer(SharedClientMessage aMessage)
{
    Shared.Log("[HOOD] [CLIENT] [NETWORK] - SendMessageToServer");
    var data = JsonConvert.SerializeObject(aMessage);
    var encoded = Encoding.UTF8.GetBytes(data);
    var buffer = new ArraySegment<Byte>(encoded, 0, encoded.Length);
    myTelepathyClient.Send(buffer);
}

```

```

public void SendMessageToPlayer(int aConnectionId, SharedServerMessage
aMessage)
{
    try
    {
        var data = JsonConvert.SerializeObject(aMessage);
        var encoded = Encoding.UTF8.GetBytes(data);
        var asWriteBuffer = new ArraySegment<Byte>(encoded, 0, encoded.Length);
        myServer.Send(aConnectionId, asWriteBuffer);
    }
    catch (Exception ex)
    {
        SharedServerMessage sem = new
        ServerInformationMessage(InformationMessageId.ERROR, ex.Message);
    }
}

```

```

        SendMessageToPlayer(aConnectionId, sem);
        ShutDownGameSession();
    }
}

public void SendMessageToAllPlayers(SharedServerMessage aMessage)
{
    foreach (KeyValuePair<int, string> playerSession in myPlayerSessionsMap)
    {
        SendMessageToPlayer(playerSession.Key, aMessage);
    }
}

private void SendMessageToOtherPlayer(int aConnectionIdToIgnore,
SharedServerMessage aMessage)
{
    foreach (KeyValuePair<int, string> playerSession in myPlayerSessionsMap)
    {
        if (playerSession.Key != aConnectionIdToIgnore)
            SendMessageToPlayer(playerSession.Key, aMessage);
    }
}

```

Para el serializado de los mensajes utilizamos las librerías `Newtonsoft.Json` y `JsonSubTypes`, esta última nos permitió la correcta serialización y deserialización posterior en base a la propiedad antes mencionada '`myType`'. Para hacer uso de la funcionalidad es imprescindible especificar los sub tipos de mensajes que necesitamos manejar mediante anotaciones sobre la definición de la clase.

```

[System.Serializable]
[JsonConverter(typeof(JsonSubtypes), "myType")]
[JsonSubtypes.KnownSubType(typeof(ClientGameplayMessage),
nameof(ClientGameplayMessage))]
[JsonSubtypes.KnownSubType(typeof(ClientPlayerGameplayMessage),
nameof(ClientPlayerGameplayMessage))]
[JsonSubtypes.KnownSubType(typeof(ClientLobbyMessage),
nameof(ClientLobbyMessage))]
[JsonSubtypes.KnownSubType(typeof(ClientMatchConnectionMessage),
nameof(ClientMatchConnectionMessage))]
[JsonSubtypes.KnownSubType(typeof(ClientPlayerReadyStatusMessage),
nameof(ClientPlayerReadyStatusMessage))]
[JsonSubtypes.KnownSubType(typeof(ClientTestFeaturesMessage),
nameof(ClientTestFeaturesMessage))]
public abstract class SharedClientMessage
{
    public abstract string myType { get; }

    public SharedClientMessage()

```

```
    }  
}  
}
```

Llamamos `OnDataReceived` a los métodos que reciben los mensajes. Estos viven uno en el servidor y otro en el cliente. Se encargan de transformar en primera instancia los bytes recibidos en `string` con formato de JSON, para luego deserializarlo en el tipo de mensaje correspondiente, `SharedClientMessage` cuando nos encontramos en el servidor y `SharedServerMessage` cuando nos encontramos en del lado del cliente. Luego se llama a una operación que se encarga de procesar el mensaje en una suerte de pasamanos, que dependiendo de la propiedad `myType` va a *castearlo* a su tipo específico y derivar el mensaje al método `ProcessMessage` correspondiente.

```
private void OnDataReceived(int aConnectionId, ArraySegment<byte> aMessage)  
{  
    try  
    {  
        Shared.Log("Data received from connectionId: " + aConnectionId);  
        string convertedMessage = Encoding.UTF8.GetString(aMessage.Array, 0,  
aMessage.Count);  
        Shared.Log("Converted message: " + convertedMessage);  
        SharedClientMessage networkMessage =  
JsonConvert.DeserializeObject<SharedClientMessage>(convertedMessage);  
        ProcessClientMessage(aConnectionId, networkMessage);  
    }  
    catch (Exception ex)  
    {  
        SharedServerMessage sem = new  
ServerInformationMessage(InformationMessageId.ERROR, ex.Message);  
        SendMessageToPlayer(aConnectionId, sem);  
        ShutDownGameSession();  
    }  
}
```

```
private void OnDataReceived(ArraySegment<byte> aMessage)  
{  
    Shared.Log("[HOOD][CLIENT][NETWORK] - OnDataReceived");  
    string convertedMessage = Encoding.UTF8.GetString(aMessage.Array, 0,  
aMessage.Count);  
    Shared.Log("[HOOD][CLIENT][NETWORK] - Converted message: " +  
convertedMessage);  
    SharedServerMessage SharedServerMessage =  
JsonConvert.DeserializeObject<SharedServerMessage>(convertedMessage);  
    ProcessServerMessage(SharedServerMessage);  
}
```

```

private void ProcessClientMessage(int aConnectionId, SharedClientMessage
aSharedClientMessage)
{
    try
    {
        switch (aSharedClientMessage.myType)
        {
            case nameof(ClientPlayerGameplayMessage):
                ClientPlayerGameplayMessage pgm = aSharedClientMessage as
ClientPlayerGameplayMessage;
                ProcessPlayerGameplayMessage(aConnectionId, pgm);
                break;
            case nameof(ClientGameplayMessage):
                ClientGameplayMessage ugm = aSharedClientMessage as
ClientGameplayMessage;
                ProcessGameplayMessage(aConnectionId, ugm);
                break;
            case nameof(ClientLobbyMessage):
                ClientLobbyMessage lm = aSharedClientMessage as
ClientLobbyMessage;
                ProcessLobbyMessage(aConnectionId, lm);
                break;
            case nameof(ClientMatchConnectionMessage):
                ClientMatchConnectionMessage mcm = aSharedClientMessage as
ClientMatchConnectionMessage;
                ProcessMatchConnectionMessage(aConnectionId, mcm);
                break;
            case nameof(ClientPlayerReadyStatusMessage):
                ClientPlayerReadyStatusMessage cprsm = aSharedClientMessage as
ClientPlayerReadyStatusMessage;
                ProcessPlayerReadyStatusMessage(aConnectionId, cprsm);
                break;
            case nameof(ClientTestFeaturesMessage):
                ClientTestFeaturesMessage ctfm = aSharedClientMessage as
ClientTestFeaturesMessage;
                ProcessClientTestFeaturesMessage(aConnectionId, ctfm);
                break;
            default:
                Shared.LogError("[HOOD][SERVER][NETWORK] - Unrecognized message
type.");
                break;
        }
    }
    catch (Exception ex)
    {
        SharedServerMessage sem = new
ServerInformationMessage(InformationMessageId.ERROR, ex.Message);
        Shared.LogError("[HOOD][SERVER][NETWORK] - ProcessClientMessage
exception. Message:" + ex.Message);
        SendMessageToPlayer(aConnectionId, sem);
        ShutDownGameSession();
    }
}

```

```
    }  
}
```

```
private void ProcessServerMessage(SharedServerMessage aSharedServerMessage)  
{  
    if (!myIsMenuSceneReferenceSet)  
    {  
        Shared.LogError("[HOOD] [CLIENT] [NETWORK] - ProcessServerMessage()")  
;  
        return;  
    }  
  
    switch (aSharedServerMessage.myType)  
    {  
        case nameof(ServerMatchSetupMessage):  
            ServerMatchSetupMessage msm = aSharedServerMessage as  
ServerMatchSetupMessage;  
            ProcessMatchSetupMessage(msm);  
            break;  
        case nameof(ServerPlayerGameplayMessage):  
            ServerPlayerGameplayMessage pgm = aSharedServerMessage as  
ServerPlayerGameplayMessage;  
            ProcessPlayerGameplayMessage(pgm);  
            break;  
        case nameof(ServerStatusGameplayMessage):  
            ServerStatusGameplayMessage sgm = aSharedServerMessage as  
ServerStatusGameplayMessage;  
            ProcessStatusGameplayMessage(sgm);  
            break;  
        case nameof(ServerGameplayActionMessage):  
            ServerGameplayActionMessage ugm = aSharedServerMessage as  
ServerGameplayActionMessage;  
            ProcessActionGameplayMessage(ugm);  
            break;  
        case nameof(ServerLobbyMessage):  
            ServerLobbyMessage lm = aSharedServerMessage as  
ServerLobbyMessage;  
            ProcessLobbyMessage(lm);  
            break;  
        case nameof(ServerReadyStatusMessage):  
            ServerReadyStatusMessage srsm = aSharedServerMessage as  
ServerReadyStatusMessage;  
            ProcessReadyStatusMessage(srsm);  
            break;  
        case nameof(ServerStartProcessMessage):  
            ServerStartProcessMessage sspm = aSharedServerMessage as  
ServerStartProcessMessage;  
            ProcessStartProcessMessage(sspm);  
            break;
```

```

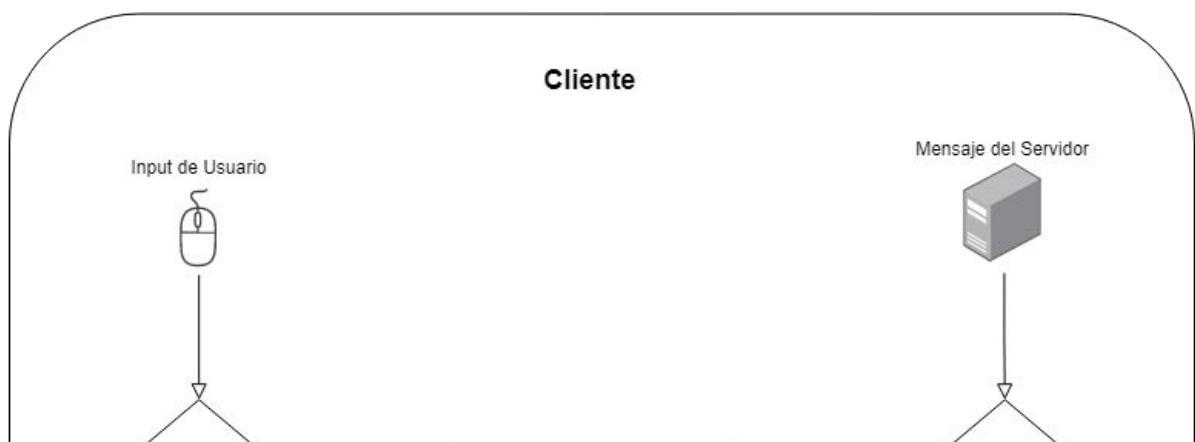
        case nameof(ServerMatchStateMessage):
            ServerMatchStateMessage smsm = aSharedServerMessage as
ServerMatchStateMessage;
            ProcessMatchStateMessage(smsm);
            break;
        case nameof(ServerInformationMessage):
            ServerInformationMessage sem = aSharedServerMessage as
ServerInformationMessage;
            ProcessErrorMessage(sem);
            break;
        case nameof(ServerDatabaseMessage):
            ServerDatabaseMessage slim = aSharedServerMessage as
ServerDatabaseMessage;
            ProcessServerDatabaseMessage(slim);
            break;
        default:
            Shared.LogError("[HOOD][CLIENT][NETWORK] - Unrecognized message
type.");
            break;
    }
}

```

### 2.8.2.1 Flujo de Acciones de Gameplay

**Duración estimada:** 2d

Con el fin de abordar las distintas acciones que un jugador puede hacer durante una partida, diseñamos un flujo de mensajes que denominamos “Request, Try, Do”. Este flujo consiste en un cliente que envía una petición de acción al servidor, luego el servidor intenta realizarla, en caso positivo procede notificar a los clientes interesados mediante otro mensaje, estos ejecutan la acción. Cabe destacar que el mensaje incluye el identificador del jugador que realizó la acción, de esta forma el cliente sabe exactamente como interpretar la información. En el caso negativo, el servidor envía un mensaje notificando que la acción no se pudo realizar. Este mensaje tendrá como destinatario solamente al jugador que realizó la petición.



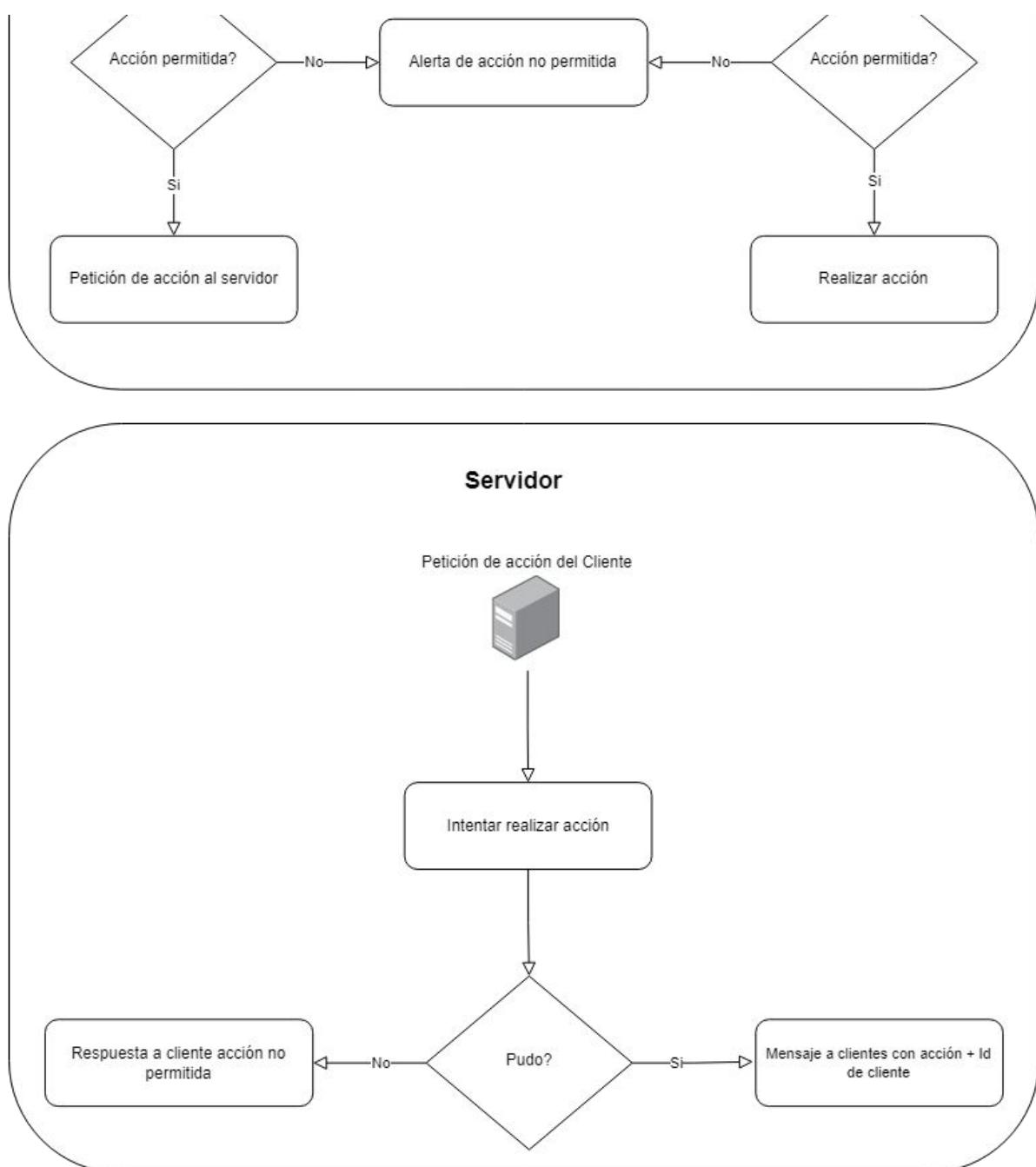


Figura 81. Flujo de acciones de gameplay

Este flujo aplica a las acciones principales de combate de un jugador durante la partida, para lo cual definimos en el código compartido dos clases de mensaje particular y dos enumerados que representan los distintos tipos de acción.

Propiedades de acciones de gameplay:

- `myMessageId` - representa la acción a realizar

- `myPlayerId` - identificador del jugador que solicita la acción
- `myGameplayObjectId` - identificador de la unidad que aplica la acción
- `myTargetPositionX` - coordenada X del tablero para obtener la celda donde se aplicará la acción
- `myTargetPositionY` - coordenada Y del tablero para obtener la celda donde se aplicará la acción

```
public class ClientGameplayMessage : SharedClientMessage
{
    public GameplayMessageIdClient myMessageId;
    public string myPlayerId;
    public int myGameplayObjectId;
    public int myTargetPositionX;
    public int myTargetPositionY;

    public ClientGameplayMessage(GameplayMessageIdClient aMessageId, string
aPlayerId, int anId, int aTargetPositionX, int aTargetPositionY) : base()
    {
        myMessageId = aMessageId;
        myPlayerId = aPlayerId;
        myGameplayObjectId = anId;
        myTargetPositionX = aTargetPositionX;
        myTargetPositionY = aTargetPositionY;
    }

    public override string myType { get; } = nameof(ClientGameplayMessage);
}
```

```
public enum GameplayMessageIdClient
{
    INVALID,
    REQUEST_SPAWN_UNIT,
    REQUEST_MOVE_UNIT,
    REQUEST_ATTACK_UNIT,
    REQUEST_USE_ABILITY
}
```

```
public class ServerGameplayActionMessage : SharedServerMessage
{
    public GameplayMessageIdServer myMessageId;
    public string myRequestingPlayerSessionId;
    public int myGameplayObjectId;
    public int myTargetPositionX;
    public int myTargetPositionY;
    public bool mySuccess;
```

```

public ServerGameplayActionMessage(GameplayMessageIdServer aMessageId,
string aRequestingPlayerSessionId, int aGameplayObjectId, int aTargetPositionX,
int aTargetPositionY, bool aSuccess) : base()
{
    myMessageId = aMessageId;
    myRequestingPlayerSessionId = aRequestingPlayerSessionId;
    myGameplayObjectId = aGameplayObjectId;
    myTargetPositionX = aTargetPositionX;
    myTargetPositionY = aTargetPositionY;
    mySuccess = aSuccess;
}

public override string myType { get; } =
nameof(ServerGameplayActionMessage);
}

```

```

public enum GameplayMessageIdServer
{
    SPAWN_UNIT,
    ATTACK_UNIT,
    MOVE_UNIT,
    USE_ABILITY
}

```

### 2.8.2.2 Cambios de Estado de Gameplay

#### Duración estimada: 1d

En una partida no sólo contamos con la posibilidad de realizar acciones de combate, sino que también desarrollamos *inputs* en forma de botón que permiten solicitar cambios de estado. Estos cambios pueden ser finalizar el turno de manera temprana, o rendirse, para que la partida culmine de forma antinatural.

Cuando el jugador presiona el botón de “finalizar turno” o “rendirse” se envía una *request* al servidor, desde donde se llama a la operación `ChangeState`. Ésta recibe un valor enumerado con el nuevo estado y el identificador del jugador que generó la *request*. A su vez hace validaciones, impacta el cambio en el servidor y envía un mensaje a todos los jugadores con el nuevo estado para que lo impacten en su cliente realizando las operaciones pertinentes.

El resto de las formas en las cuales el estado de la partida debe cambiar es ordenado por el servidor directamente. Por ejemplo, cuando finaliza el *timer* del turno o cuando se cumple con la condición de victoria. En cualquiera de estos casos el servidor se encarga de notificar mediante un mensaje a todos los jugadores de la partida y estos se encargan de interpretarlo.

### 2.8.2.3 Información del Oponente

#### Duración estimada: 1d

Para el momento de un cliente unirse a una partida, más precisamente al *lobby*, debemos manejar cuidadosamente el manejo de la información del oponente. Identificamos distintos momentos en los cuales se debería actualizar el panel del enemigo. Cuando un cliente es el *host* de la partida y se une un oponente, debemos mostrar los datos en pantalla. También, cuando este cliente se une debe recibir la información del jugador que ya se encontraba dentro de ella. Todos estos datos deben ser vaciados una vez corresponda según el flujo de desconexión de una partida.

### 2.8.2.4 Flujo de Desconexión

#### Duración estimada: 1d

Un cliente se encuentra en estado conectado una vez que crea o se une a una partida, pero tiene la posibilidad de desconectarse en casi todo momento. Definimos distintos comportamientos y cierres de partida dependiendo del momento de desconexión.

#### Desconexión durante lobby

- Host - se cierra la partida para todos los involucrados.
- Guest - cierra la partida solamente para él.
- Cierre de servidor - ambos jugadores vuelven al menú principal.

#### Desconexión durante gameplay

- Se da como ganador al jugador que quede en pie.
- Cierre de servidor - ambos jugadores vuelven al menú principal.

## 2.9 Finalización 3

### 2.9.1 Administración de Tareas con Contingencias

#### Duración estimada: n/a

Llegados los últimos días de desarrollo, en un esfuerzo de reducir riesgos, adoptamos un sistema más estricto de control de tareas usando un diagrama temporal.

Para representar didácticamente esta herramienta iremos ajustando un diagrama de ejemplo a medida que explicamos los puntos individuales. Este sistema cuenta con las siguientes propiedades:

### 2.9.1.1 Tiempo

Generar tareas unitarias con una estimación de tiempo simple, éstas se ven representadas en el tiempo en relación a los días restantes.

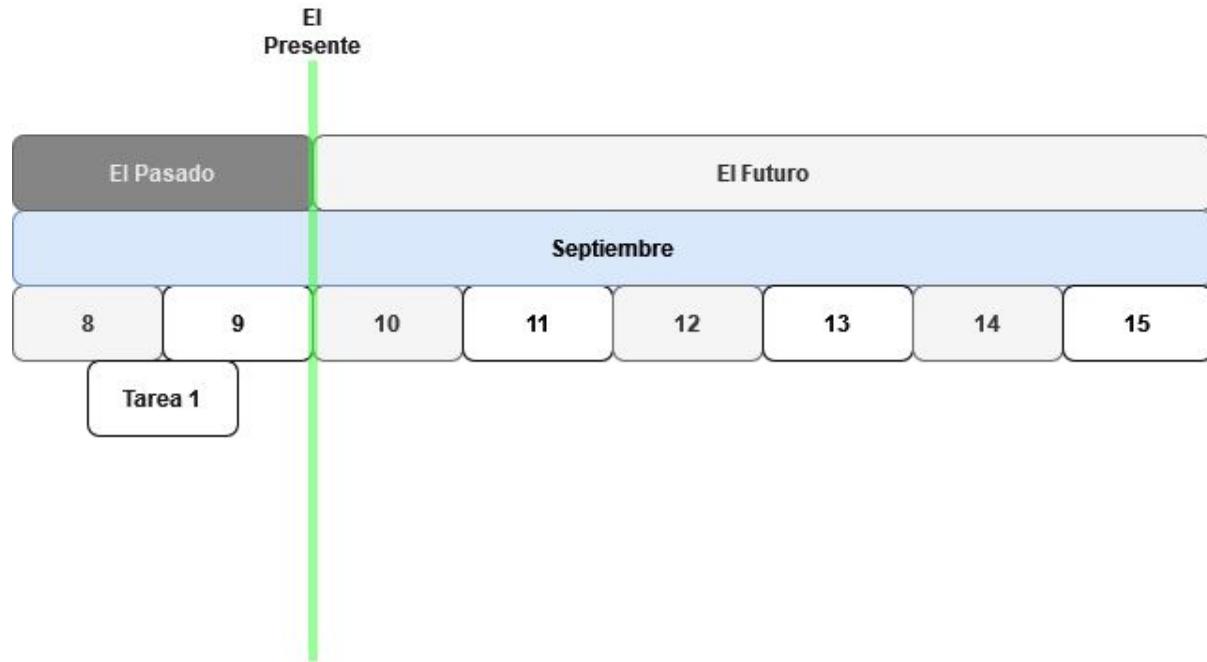


Figura 82. Administración de tareas con contingencias, tiempo

### 2.9.1.2 Orden

Las tareas deben identificar de manera clara cuáles son las otras tareas deben cumplirse para que sea posible iniciarlas, una tarea que bloquea a otra se representa en la misma línea, con la tarea que bloquea a la izquierda de la que es bloqueada. En caso de ser posible parallelizar las tareas, se colocan en líneas paralelas.

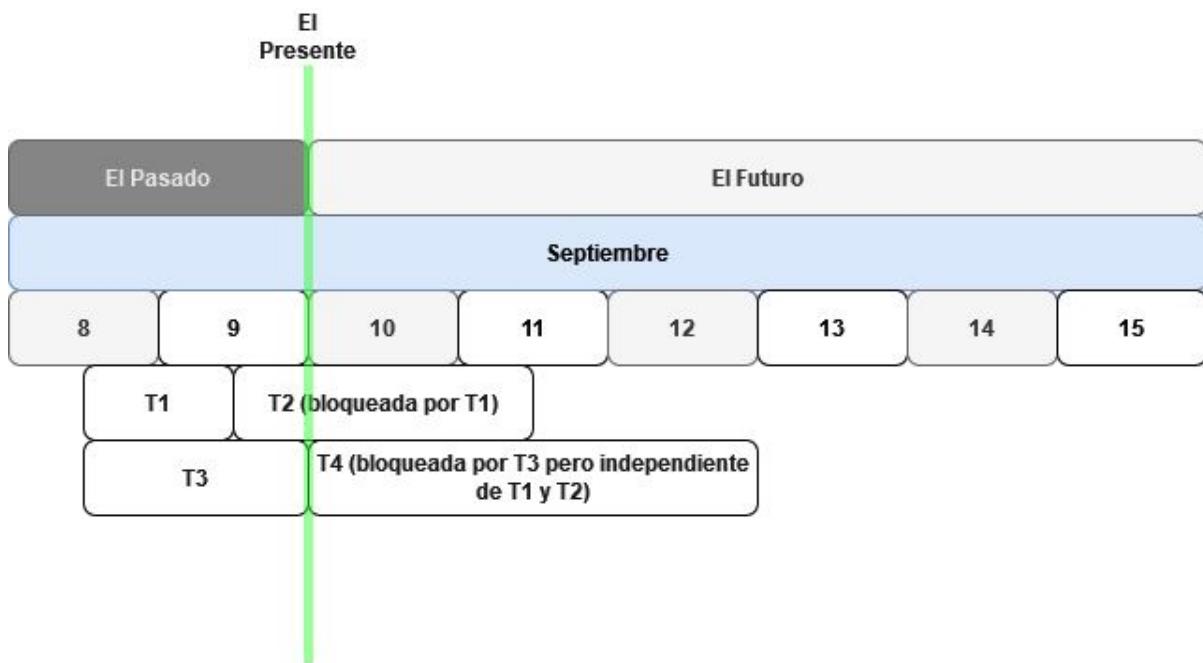


Figura 83. Administración de tareas con contingencias, orden

### 2.9.1.3 Prioridad

Algunas tareas están marcadas como pertenecientes al camino crítico para llegar al mínimo producto final aceptable. Marcamos éstas con rojo, tareas menos prioritarias con amarillo y tareas prescindibles con verde.

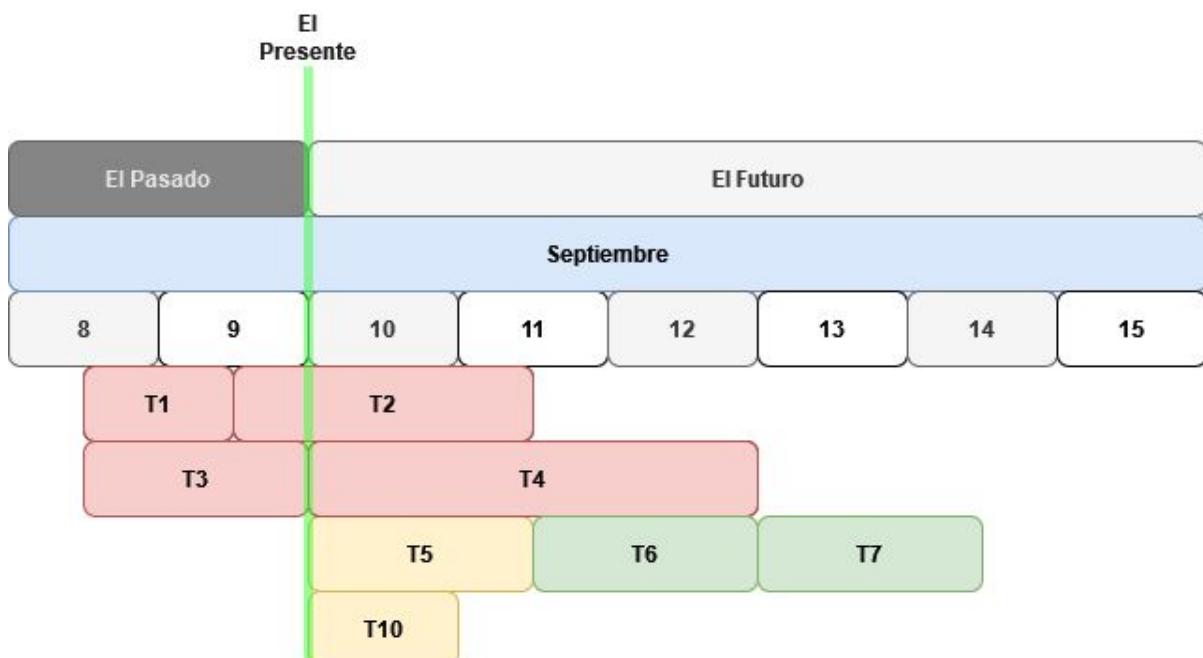


Figura 84. Administración de tareas con contingencias, prioridad

#### 2.9.1.4 Contingencias

Se eligen varios puntos de control que agrupan tareas pertenecientes al camino crítico, de no llegar a uno de estos puntos de control con todas las tareas de su grupo cumplida, se eliminan tareas prescindibles marcándolas con negro, recuperando de esta manera tiempo de desarrollo. Si esto ocurre se re-evalúan las prioridades de las tareas restantes. Las tareas completas se marcan con magenta.

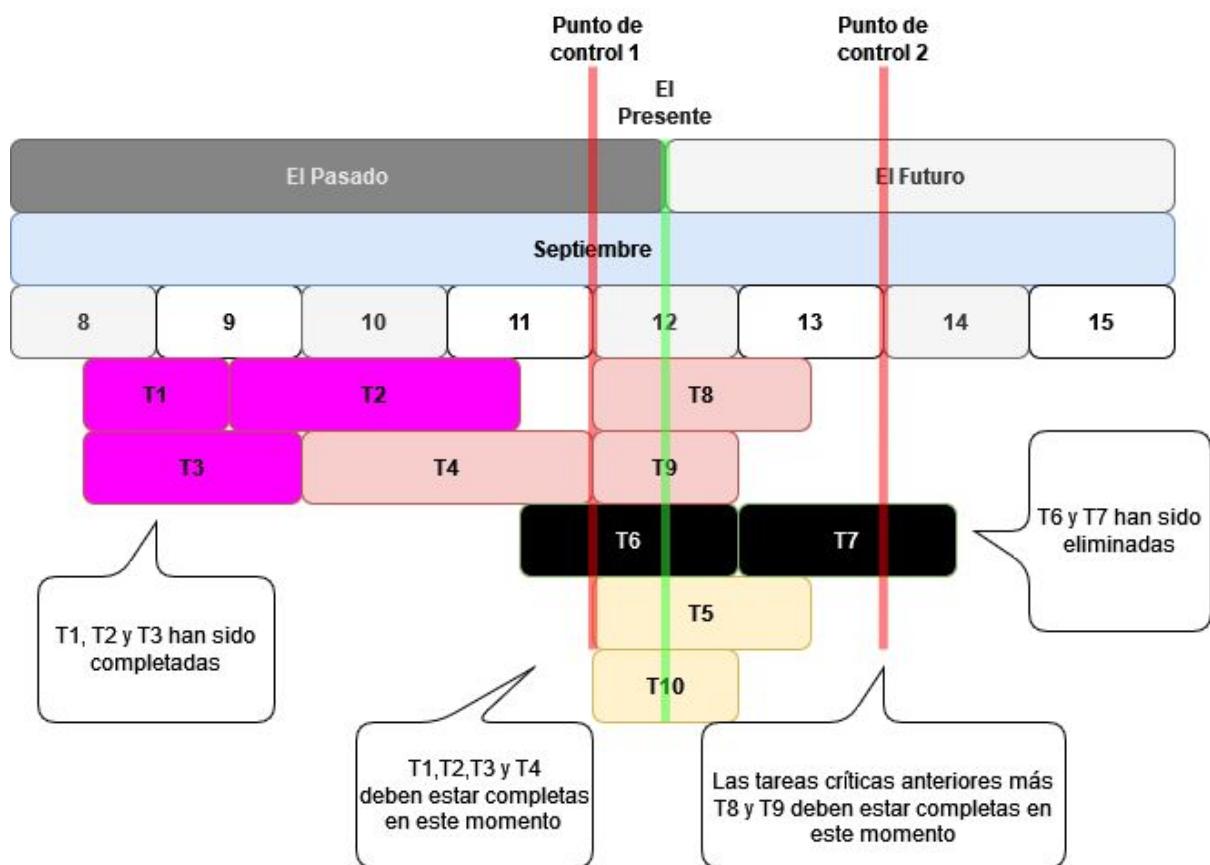


Figura 85. Administración de tareas con contingencias, contingencias

#### 2.9.2 Command Line Utility

**Duración estimada:** 2d

A raíz de manejar configuraciones cambiantes o múltiples sesiones de usuario a la hora de verificar funcionalidades en la aplicación del cliente, se hizo evidente la necesidad de poder configurar el flujo del código de manera flexible y dinámica para poder desarrollar de manera efectiva.

La solución propuesta fue utilizar argumentos de consola que la aplicación interpreta al iniciar, consultando estos valores en el código podemos modificar el flujo del juego sin necesidad de compilaciones diferenciadas.

Partimos de la herramienta desarrollada como código libre para Unity y disponible en GitHub, Unity Command Line [v20] esta nos permite capturar valores de tipo *boolean*, *integer*, *string* y *enum* al inicializar un proyecto de Unity. Los valores se pueden leer de un documento de texto plano o enviarlos por linea de comando al ejecutar la aplicación.

Por ejemplo, en el caso de necesitar un valor tipo *string* que contenga el identificador actual de la *fleet* de GameLift activa.

### 2.9.2.1 Declaración del Argumento

En un documento de texto llamado `ClientArguments.txt` ubicado en la carpeta `./Assets/StreamingAssets`

```
-FLEET_ID "fleet-fcc70ae4-577b-496b-a9cc-b463be8e7504"
```

### 2.9.2.2 Captura del Argumento

```
using UnityEngine;
using Oddworm.Framework;
public class CLU : MonoBehaviour
{
    public static string GetFleetId()
    {
        // Si el argumento no es encontrado devolvemos ""
        return CommandLine.GetString("-FLEET_ID", "");
    }

    [RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.AfterAssembliesLoaded)]
    static void LoadCommandLine()
    {
        // Usar argumentos pasados a la aplicación directamente
        var text = System.Environment.CommandLine + "\n";

        // Usar argumentos declarados en el archivo de texto plano
        var path = System.IO.Path.Combine(Application.streamingAssetsPath,
"ClientArguments.txt");
```

```

    if (System.IO.File.Exists(path))
    {
        text += System.IO.File.ReadAllText(path);
    }
    else
    {
        #if UNITY_EDITOR || USE_ARGUMENTS
            Shared.LogError("[HOOD][CLU] Could not find commandline file: " +
path);
        #endif
    }

    // Inicializar CommandLine
    Oddworm.Framework.CommandLine.Init(text);
}
}

```

### 2.9.2.3 Uso del Argumento

Desde cualquier parte del proyecto se puede usar esta información

```

public class GameLiftClient : MonoBehaviour
{
    //...
    async private Task<GameSession> SearchPrivateGameSessionAsync(string
aGameSessionId)
    {
        //...
        searchGameSessionsRequest.FleetId = CLU.GetFleetId();
        //...
    }
    //...
}

```

### 2.9.3 Local Game Server

**Duración estimada:** 2d

En las últimas etapas de desarrollo, cuando la velocidad iteración del equipo es más alta y aumentaba la complejidad de la lógica agregada a nuestro servidor, se volvió evidente que un factor que impedía el avance era la necesidad de validar la lógica de servidor en un servidor remoto de AWS GameLift.

### 2.9.3.1 Flujo de validación en un servidor remoto de AWS GameLift

1. Hay una nueva funcionalidad de servidor para validar
2. Borrar *fleet* en AWS, 10 minutos promedio
3. Crear y levantar una nueva *fleet* en AWS, 17 minutos promedio
4. Compilar un cliente con Unity usando los datos de la nueva *fleet*, 5 minutos promedio
5. Validar la funcionalidad

Si el código de servidor encuentra una excepción que impide cerrar las sesiones de juego activas, se puede incurrir en unos 6 minutos adicionales para reiniciar las instancias, o 10 minutos para borrar la *fleet*. En total entre 38 y 42 minutos para validar cada nueva funcionalidad agregada a la lógica del servidor.

### 2.9.3.2 Alternativas

Teniendo en cuenta que la interacción de la API de GameLift con un cliente de Unity es sólo necesaria al momento de crear sesiones de juego y con el servidor de Unity al momento de terminar sesiones de juego.

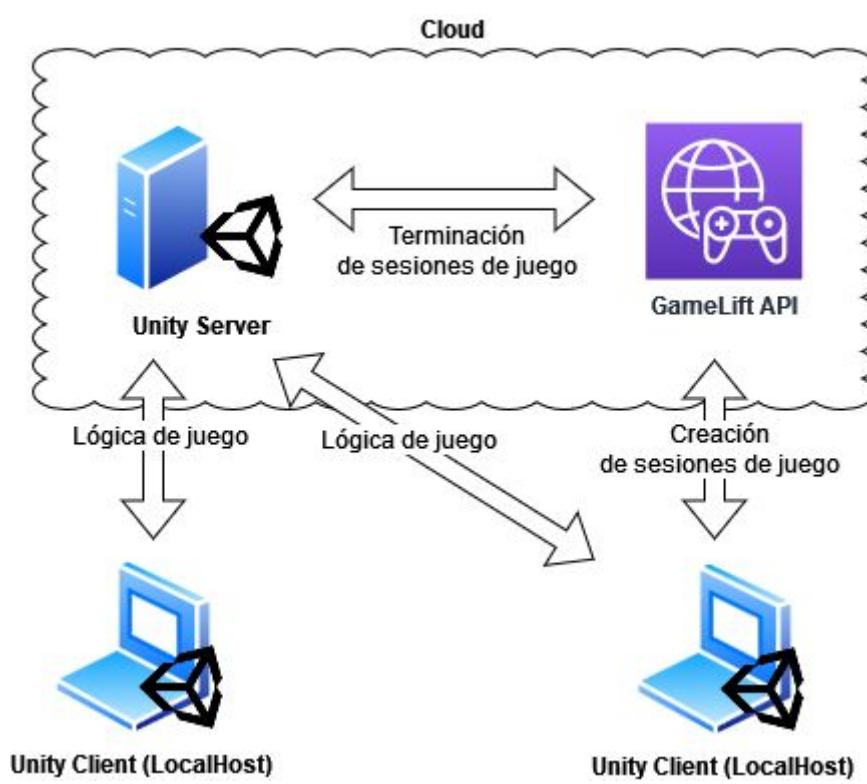


Figura 86. Interacción de clientes con servidor remoto usando AWS GameLift

Es posible, para funcionalidades que no incluyan la creación o terminación de sesiones de juego, validar funcionalidades de servidor enteramente en el dispositivo de un desarrollador.

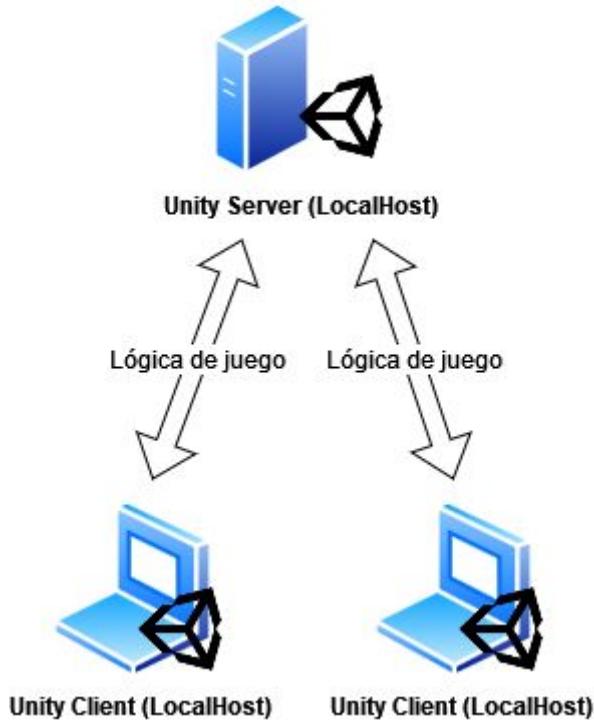


Figura 87. Interacción de clientes con servidor local

Este cambio probó ser de inmensa utilidad ya que provee una herramienta invaluable al dar la posibilidad de iterar instantáneamente lógica de *gameplay* que se desarrolla enteramente en el servidor de Unity, acelerando tiempos y la independencia de los desarrolladores.

## 2.9.4 Manejo de Excepciones en Servidor

**Duración estimada:** 1d

Cuando comenzamos a probar funcionalidades también comenzaron las excepciones no controladas en el servidor generadas por errores en la programación. Esto nos insumía una cantidad de tiempo considerable, sobre todo al momento que no contábamos con entorno de *testing* local porque implicaba que el servidor dejara de responder a los mensajes y debíamos volver a levantarla para realizar otra prueba. Por este motivo es que surgimos con la idea de manejar las excepciones enviando un mensaje al cliente para poder visualizar lo que estaba ocurriendo mediante un *log*. En ese momento devolvemos el servidor a un estado sano y sacamos a los jugadores de la partida.

#### 2.9.4.1 Enumerado de tipos de errores

```
public enum InformationMessageId
{
    ERROR,
    INFO,
    SERVER_CLOSED
}
```

#### 2.9.4.2 Clase que transporta el mensaje de error

```
public class ServerInformationMessage : SharedServerMessage
{
    public InformationMessageId myMessageId;
    public string myMessage;

    public ServerInformationMessage(InformationMessageId aMessageId, string
aError) : base()
    {
        myMessageId = aMessageId;
        myMessage = aError;
    }

    public override string myType { get; } =
nameof(ServerInformationMessage);
}
```

### 2.9.5 Evolución del Aspecto Visual del Juego

**Duración estimada:** 2d

Para mejorar la comprensibilidad del juego, se optó por mejorar algunos aspectos visuales de la partida.

#### 2.9.5.1 Unidades

Con la creación de las nuevas unidades “Fenix”, “Hunter”, “Rocket”, “Tank” y “Traveler”, se crearon animaciones en reposo para cada una de ellas. De esta forma, los distintos tipos de unidades serían reconocibles a la vista para los jugadores. Estos sprites fueron obtenidos del Unity Asset Store [v24]



Figura 88. Sprite de “Fenix”



Figura 89. Sprite de “Rocket”



Figura 90. Sprite de “Tank”



Figura 91. Sprite de “Hunter”



Figura 92. Sprite de “Traveler”

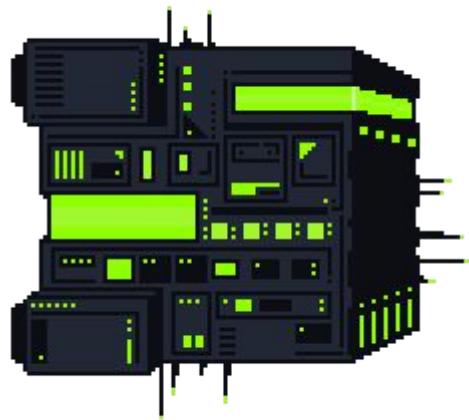


Figura 93. Nuevo sprite de la nave nodriza.

#### 2.9.5.2 Adicionalmente

- Las unidades que surgen del jugador que comienza la partida del lado derecho del tablero se ven espejadas.
- Las unidades que corresponden a un jugador rival, tienen la palabra “Enemy” por encima de ellas.

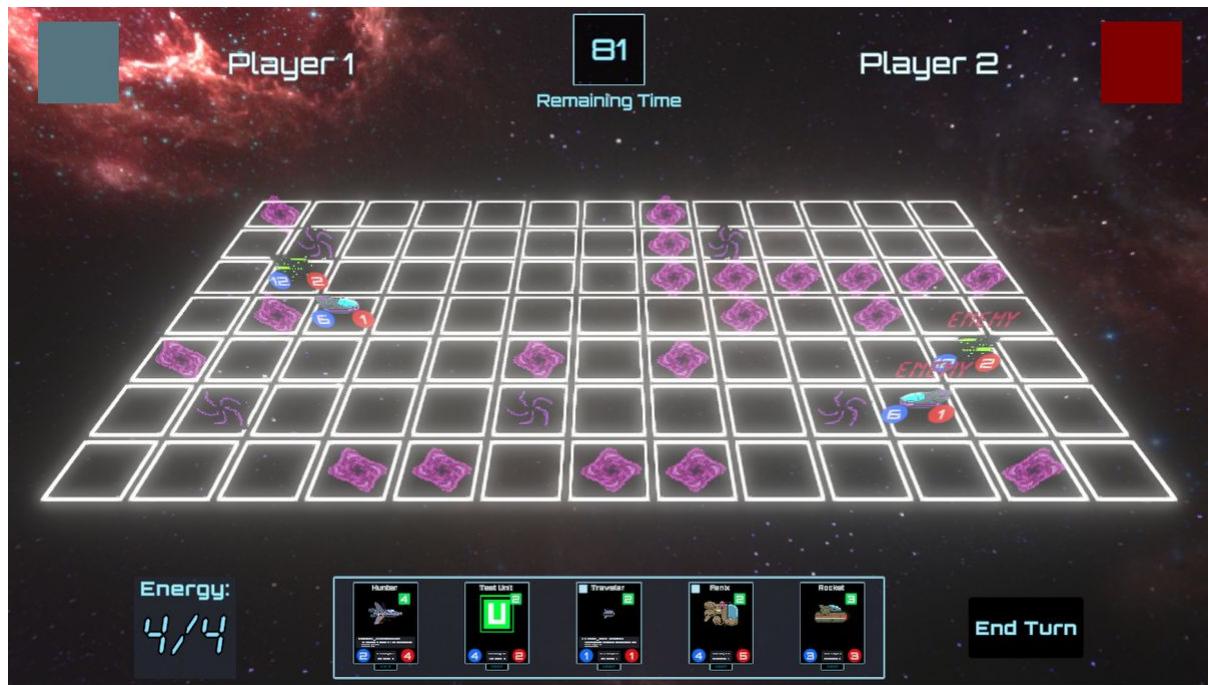


Figura 94. Tablero con mejoras visuales

### 2.9.5.3 Interfaz de Usuario

Un agregado a la interfaz de usuario consta de la capacidad de un jugador de mantener su mouse sobre una unidad, y que esto muestre una carta con los datos de esa unidad. Esto permite a los usuarios a saber qué atributos tiene una unidad sin ser necesario aprenderlos de memoria.

También se agregó que los nombres de los jugadores aparezcan en pantalla durante una partida.

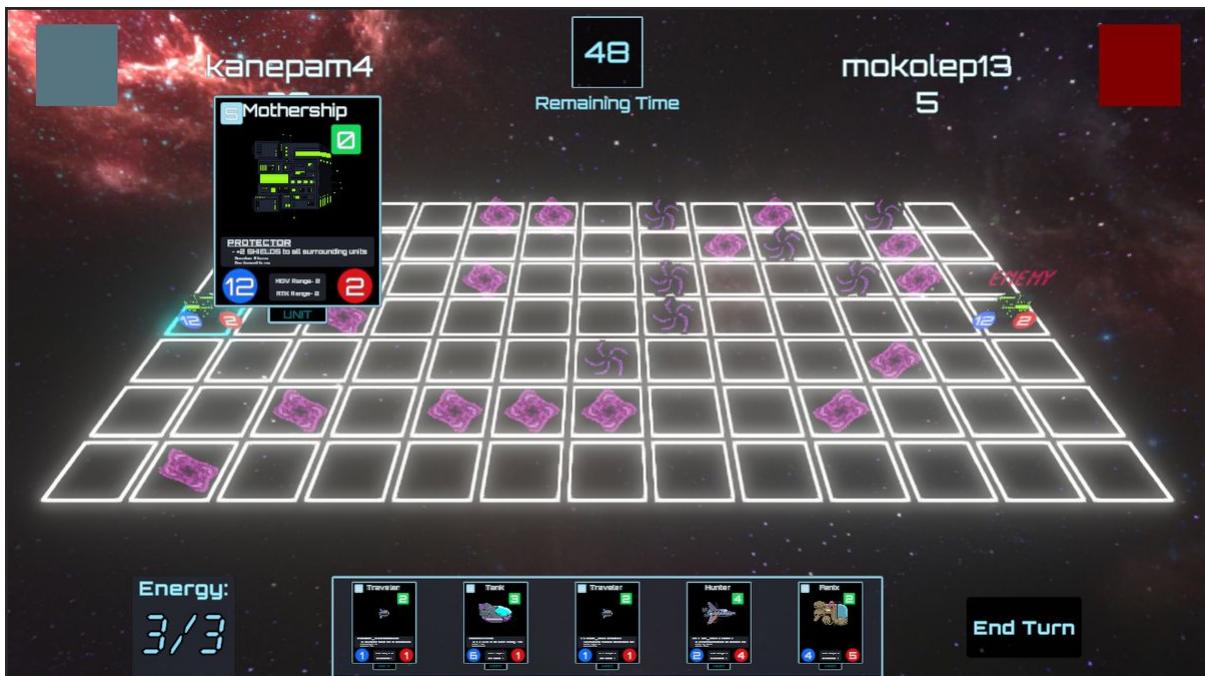


Figura 95. Tablero mostrando

### 2.9.6 Sincronización Servidor-Cliente de los Datos Iniciales de una Partida

Nuestra arquitectura de las partidas se basa en que el servidor funcione como enlace y juez entre ambos clientes. Un jugador hace una acción, el servidor la confirma y le transmite el resultado de esta acción a ambos jugadores.

Sin embargo, para que este sistema efectivamente sincronice a ambos jugadores con el servidor y entre ellos, se debe partir del mismo punto de partida. Por esta razón, se debió ingeniar una manera de hacer consistentes a los elementos aleatorios de las partidas.

### 2.9.6.1 Algoritmo de Fisher-Yates

El algoritmo de Fisher-Yates es un algoritmo de permutaciones que permite desordenar una lista de forma aleatoria, y en una implementación que incorpora que incorpora una *seed*, es capaz de desordenar una lista de forma predecible [2].

```
public T[] ShuffleItemsWithSeed<T>(T[] anArray, int aSeed)
{
    System.Random randomNumber = new System.Random(aSeed);

    for (int i = 0; i < anArray.Length - 1; i++)
    {
        int randomIndex = randomNumber.Next(i, anArray.Count());
        T tempItem = anArray[randomIndex];

        anArray[randomIndex] = anArray[i];
        anArray[i] = tempItem;
    }
    return anArray;
}
```

Usando el método `Next()` la clase nativa de C# `Random(int seed)` somos capaces de siempre obtener la misma secuencia de números pseudo-aleatorios para nuestro algoritmo. En otras palabras, si se provee el mismo *array* y el mismo *seed* al método, siempre se devolverá el mismo *array* (des)ordenado de forma idéntica.

### 2.9.6.2 Tablero

En un principio, el tablero decidía celda por celda cuál sería su tipo de celda final, basándose en porcentajes usando el siguiente método:

```
/private TileType GetRandomTileType()
{
    int randomNum = new System.Random(mySeed).Next(1, 100);

    switch (randomNum)
    {
        case <= 10:
            return TileType.BLACKHOLE; // 10 % chance of BH
        case <= 20:
            return TileType.NEBULA; // 10 % chance of NB
        default:
            return TileType.EMPTY; // 80 % chance of empty
    }
}
```

```
}
```

Sin embargo, este método crea un tablero con distintas cantidades y posiciones de los tipos de celdas para el server y para cada uno de los clientes. Fue necesario modificar la forma en la que se crea el tablero para poder tener un juego en línea.

Como fue mencionado, el punto de partida tiene que ser el mismo para que el algoritmo produzca los mismos resultados. Entonces, los tipos de celdas no debían ser decididos por celda, sino que se debía partir de un número fijo de celdas especiales.

El servidor guarda un número aleatorio de celdas de nébula y luego hace lo mismo para las celdas de agujero negro, de esta manera se le pueden proveer a los jugadores en un futuro, cuando creen sus propios tableros. Lo mismo sucede con la *seed*, que se también se guarda.

Luego, estas celdas especiales se agregan a una lista una por una hasta agotar su cantidad. El producto siendo una lista con celdas nébula, seguidas por celdas agujero negro, seguidas por celdas vacías. En este momento del proceso se tiene el punto de partida idéntico para todos los actores y que se está en condiciones para mezclar la lista con el *seed* mencionado.

```
private void GenerateGrid()
{
    if (!IsDataLoaded() || myBlackHoleQty < 0 || myNebulaQty < 0)
    {
        Shared.LogError("[HOOD] [BOARD] [ERROR] - GenerateGrid()");
        return;
    }

    int nebulaToPlace = myNebulaQty;
    int blackHoleToPlace = myBlackHoleQty;
    List<TileType> tileTypeList = new();

    for (int i = 0; i < myWidth * myHeight; i++)
    {
        TileType tileType = SelectTile(nebulaToPlace, blackHoleToPlace);

        if (tileType == TileType.NEBULA)
            nebulaToPlace--;
        if (tileType == TileType.BLACKHOLE)
            blackHoleToPlace--;

        tileTypeList.Add(tileType);
    }

    tileTypeList =
        myGameManagerReference.ShuffleItemsWithSeed(tileTypeList.ToArray(),
        mySeed).ToList();
}
```

```
        InstantiateBoardWithCoords(ConvertToListList(tileTypeList));
    }
```

Una vez mezclada la lista, se le provee a cada celda de su coordenada para que cumpla su función en el juego.

Antes de enviar los datos a los clientes para que hagan el mismo proceso, el server se asegura que el tablero creado sea jugable, es decir, que todas las celdas sean accesibles y que no hayan quedado porciones del mapa inaccesibles o más grave, que ambos jugadores queden aislados entre sí por una columna de agujeros negros impasables. En caso de que esto sea así, el servidor cambia de *seed* y intenta crear el tablero nuevamente.

Esos se consigue con el método `IsMapFullyAccessible` que utiliza funciones ya programadas para cumplir con su función.

En nuestro juego ya existe una forma de determinar a qué celdas se puede mover una unidad, determinado por su rango de movimiento. Es el caso, entonces, que una unidad con muy alto rango de movimiento podrá acceder a todas las celdas del mapa que no sean impasables.

Esto llevó a la solución donde el servidor crea una unidad auxiliar con esta característica y compara que las celdas en donde tiene permitido moverse sean las mismas que todas las celdas no impasables del tablero. Si lo son, el tablero es completamente accesible y aceptable para una partida.

### 2.9.6.3 Mazo

El mazo de cada cliente tiene que ser conocido por el servidor, ya que tiene que decidir si un usuario debería tener determinada carta en su mano en determinado momento, para evitar trampas. Entonces, el mazo debe mezclarse de la misma manera para el servidor y para el cliente.

Como las posiciones iniciales de las cartas dentro de un mazo son pre-definidas, el proceso de mezcla descripto anteriormente siempre las mezcla de la misma manera y el juego puede fluir de forma correcta.

### 3 Evolución de Stack de Tecnologías

#### 3.1 Versión 1

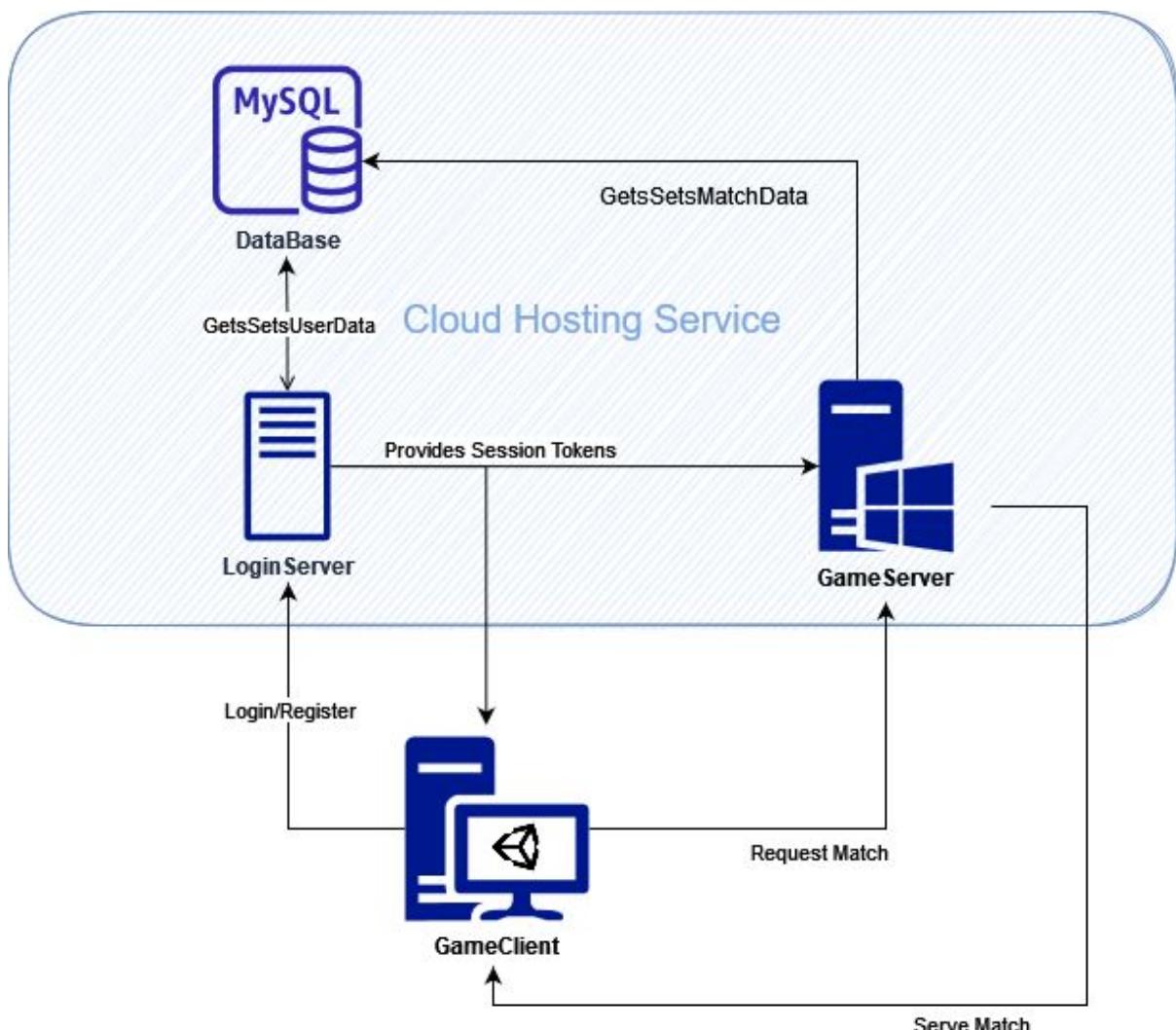


Figura 96. Stack de tecnologías v01

### 3.2 Versión 2

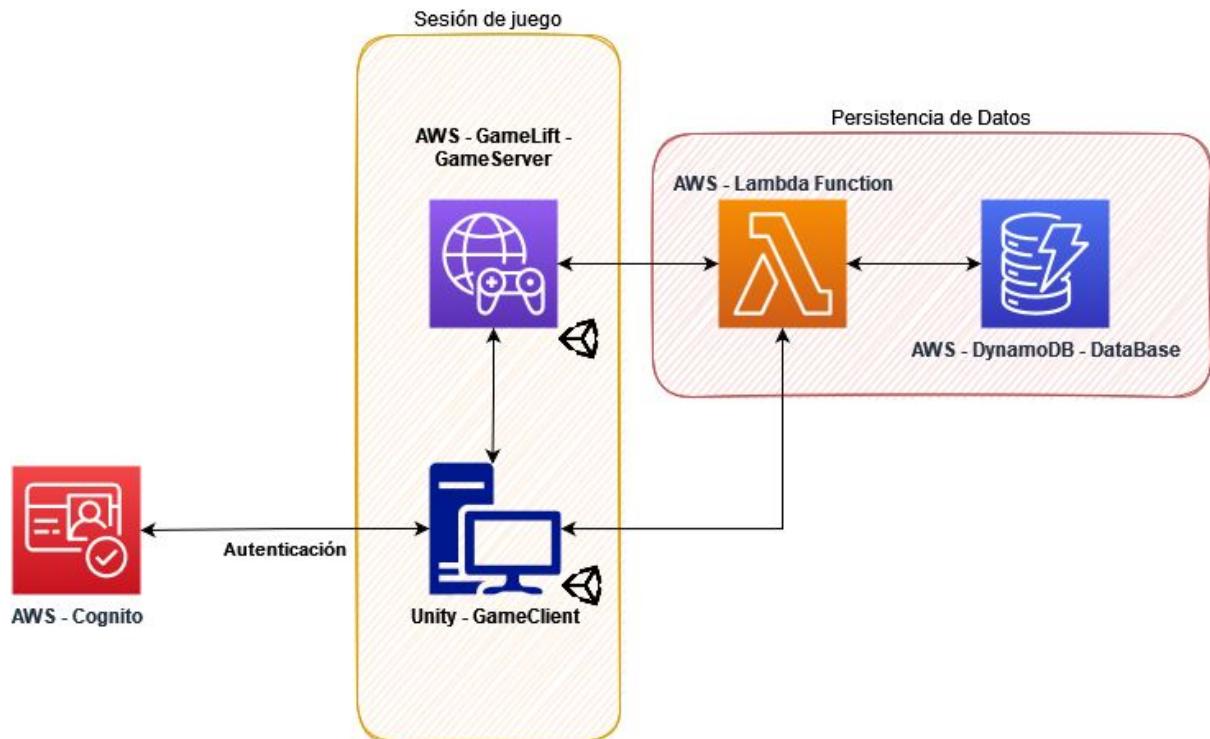


Figura 97. Stack de tecnologías v02

## 4 Conclusiones

### 4.1 Introducción

En este capítulo presentamos el producto final de este esfuerzo de equipo y evaluaremos como se compara con los compromisos y metas que nos planteamos al inicio.

#### 4.1.1 Meta Inicial

Parte de nuestra meta inicial se puede deducir desde nuestra [propuesta generada en la etapa de anteproyecto](#).

##### 4.1.1.1 Aspectos de Propuesta Inicial

- Juego competitivo en línea 1 vs 1, alcanzado
- Componentes de estrategia, alcanzado
- Área de juego con grilla cuadriculada, alcanzado
- Mazos personalizables, omitido
- Cartas que generan acciones diferenciadas, alcanzado
- Unidades que existen en el tablero con diferentes posibles interacciones, alcanzado

#### 4.1.2 Producto Final

Nuestro producto final cumple por lo menos con 5 de esos 6 aspectos. Sin embargo, todos fueron considerados e incluidos en el desarrollo y otros fueron sumados durante el desarrollo.

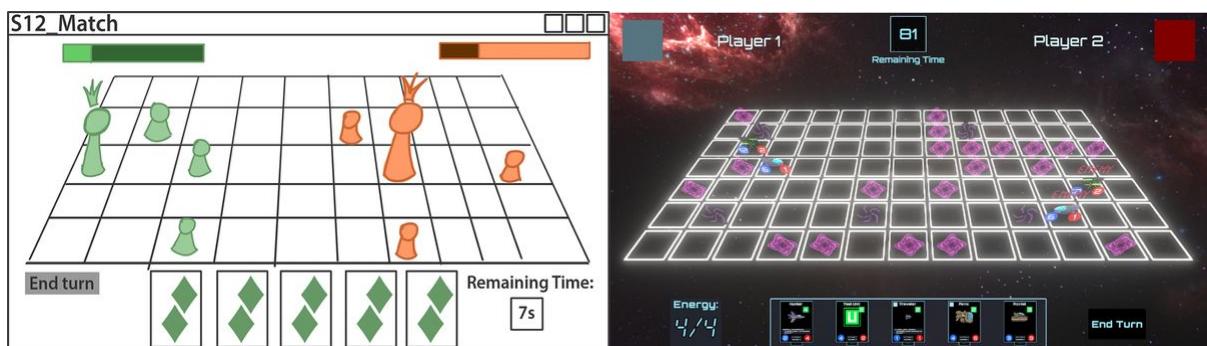


Figura 98. Izquierda diagrama inicial, derecha versión actual.

##### 4.1.2.1 Algunas Métricas del Camino Recorrido Incluyen

- 10334 líneas de código en C#

- 5 Proyectos de Visual Studio
- 5 Servicios de AWS
- 4 Repositorios de código
- 160 Commits al software de control de versión (entre varios repositorios)
- 4034 Mensajes de Discord

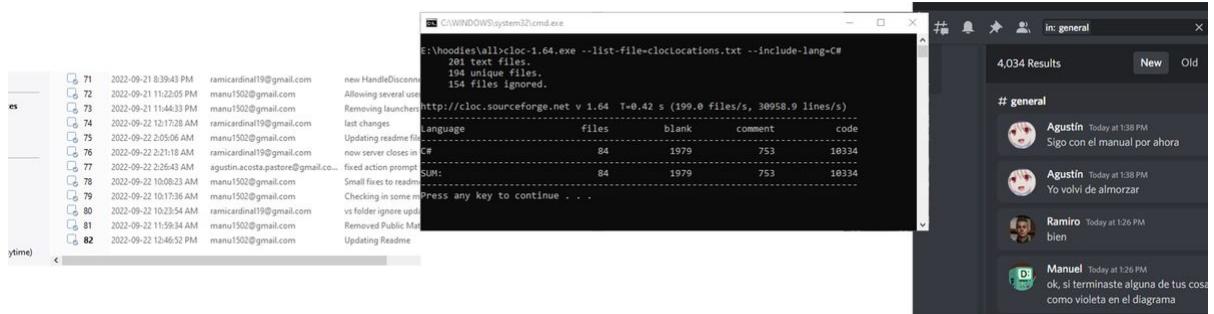


Figura 99. Algunas métricas.

Estamos bastante satisfechos con nuestro desempeño y creemos que hemos producido un resultado de valor que se ajusta con nuestras intenciones iniciales.

## 4.2 Requerimientos

En esta sección contrastamos los requerimientos planteados contra los requerimientos alcanzados. Algunos fueron alcanzados, otros no fueron realizados en su completitud o sufrieron cambios justificados, y otros fueron descartados por razones mayores. Hubo casos particulares en dónde la prioridad fue alterada en el transcurso del proyecto.

### 4.2.1 Requerimientos Funcionales

<b>Id</b>	<b>Nombre</b>	<b>Resumen</b>	<b>Estado</b>
RF-1	S01_Login	El requerimiento fue alcanzado pero sufrió modificaciones. La pantalla en la que se muestra fue combinada con la pantalla esperada en el requerimiento funcional RF-2	Completo con modificaciones

RF-2	S02_Register	El requerimiento fue alcanzado pero sufrió modificaciones. La pantalla en la que se muestra fue combinada con la pantalla esperada en el requerimiento funcional RF-1	Completo con modificaciones
RF-3	S11_AccountRecovery	El requerimiento no fue alcanzado. Su prioridad era muy baja por lo que fue descartado para lograr tareas de mayor prioridad.	Descartado
RF-4	S03_MainMenu	El requerimiento fue alcanzado en su completitud.	Completo
RF-5	S05_PlayMenu	El requerimiento fue alcanzado parcialmente, el botón public match fue ocultado por falta de tiempo para testing	Completo parcialmente
RF_6	S04_Decks	El requerimiento no fue alcanzado. Si bien realizamos toda la interfaz necesaria, hubo que descartar el requerimiento para cumplir con el camino crítico del proyecto.	Descartado
RF_7	S06_SetPrivateMatch	El requerimiento fue alcanzado en su completitud. Tomó un rol importante durante el proyecto transformándose en un requerimiento de mayor prioridad para cumplir con el camino crítico.	Completo

RF_8	S07_PrivateMatchLobby	El requerimiento fue alcanzado, pero descartamos la opción de mostrar la lista de decks en un combo desplegable al momento de descartar el requerimiento funcional RF-6.	Completo parcialmente
RF_9	S08_PublicMatchmaking	El requerimiento no fue alcanzado. Si bien la funcionalidad fue implementada, debimos descartarla por falta de testing. No aportaba al camino crítico del proyecto.	Descartado
RF_10	S09_PublicMatchLobby	El requerimiento no fue alcanzado. Si bien la funcionalidad fue implementada, debimos descartarla por falta de testing. No aportaba al camino crítico del proyecto.	Descartado
RF_11	S10_MatchReady	El requerimiento fue alcanzado en su completitud.	Completo
RF_12	S12_Match	El requerimiento fue alcanzado en su completitud.	Completo
RF_13	D01_LoginServer	El requerimiento fue alcanzado en su completitud.	Completo
RF_14	D02_Database	El requerimiento fue alcanzado parcialmente, contamos con una base de datos y la usamos activamente pero no es utilizada con la extensión originalmente planteada.	Completo parcialmente
RF_15	D03_GameServer	El requerimiento fue alcanzado en su completitud.	Completo

RF_16	S13_EndMatch	El requerimiento fue alcanzado en su completitud.	Completo
RF_17	S14_Settings	El requerimiento fue alcanzado en su completitud.	Completo

Tabla 22. Estado de requerimientos funcionales.

#### 4.2.2 Requerimientos No Funcionales

<b>Id</b>	<b>Resumen</b>	<b>Estado</b>
RNF_1	El requerimiento fue alcanzado en su completitud.	Completo
RNF_2	El requerimiento fue alcanzado en su completitud. Destacamos que no se encuentra activo en todo momento debido a temas de costo de AWS.	Completo
RNF_3	El requerimiento fue alcanzado en su completitud.	Completo
RNF_4	El requerimiento fue alcanzado en su completitud.	Completo
RNF_5	El requerimiento no fue alcanzado.	Descartado
RNF_6	El requerimiento fue alcanzado en su completitud.	Completo
RNF_7	El requerimiento fue alcanzado parcialmente, no todos los métodos cuentan con su explicación en formato de comentario.	Completo parcialmente
RNF_8	El requerimiento fue alcanzado en su completitud.	Completo

Tabla 23. Estado de requerimientos no funcionales.

### 4.3 Riesgos

La inexperiencia en proyectos de esta magnitud y teniendo en cuenta la complejidad de la naturaleza del proyecto que elegimos, hace incluso que los riesgos sean difíciles de

prever. De todas formas, los riesgos planteados en el anteproyecto fueron acertados, lo cuál no implica que la estrategia de mitigación planteada haya sido la correcta.

### 4.3.1 Manifestación de Riesgos

#### 4.3.1.1 R01 - Necesidad de uso de nuevas tecnologías y herramientas

Considerábamos que éste era un riesgo de alto impacto que fue reducido levemente aplicando la estrategia planteada de ayudarnos mutuamente compartiendo la experiencia de cada uno y con contacto constante mediante la herramienta de comunicación Discord. Se debió dedicar mucho tiempo a la investigación y aprendizaje.

#### 4.3.1.2 R02 - Mala estimación de los tiempos de desarrollo

Riesgo considerado muy común que aplicó en nuestro proyecto, debimos re-estimar en muchos momentos e incluso modificar objetivos del proyecto como fue planteado en la estrategia de mitigación. De todas formas consideramos que tuvo un impacto en el alcance del proyecto.

#### 4.3.1.3 R03 - Error en elección de la herramienta a usar

En ciertos casos ocurrió que determinamos que la herramienta elegida no había sido la correcta, lo cual significó un impacto considerable en el alcance del proyecto que pudo haber sido mayor en caso de no haber previsto alternativas.

#### 4.3.1.4 R04 - In-disponibilidad de alguno/s de los integrantes

Si bien algunos integrantes realizaron viajes o estuvieron indiscretos temas personales y pérdidas de seres queridos, los tiempos que estuvieron indiscretos fueron muy cortos y no significó una amenaza para el proyecto.

A modo de retrospectiva, algunos de los riesgos podrían haber sido planteados con mayor rigurosidad en cuanto al impacto.

### 4.3.2 Grupo Foco

Durante la introducción de nuestro proyecto fue mencionado que el mismo surge de motivación propia, por lo cual buscamos una contrapartida para reemplazar la figura del cliente que suele ser estándar en proyectos de carrera, esta figura es la de **Grupo Foco**.

Si bien inicialmente tuvimos un contacto positivo con los miembros del principal grupo foco, este eventualmente presentó problemas para cumplir con su parte de la comunicación y tuvimos que concluir esa interacción. Mientras que teníamos a nuestro alcance alternativas, este cambio nos encuentra justo en un momento previo a grandes demoras en el desarrollo, por lo cual no producir material apropiado para presentar ante un nuevo grupo foco, con el pasar de los *sprints* esto perdió prioridad en relación a cumplir con requerimientos vitales.

## 4.4 Herramientas y Tecnologías

Algunas [herramientas que elegimos inicialmente](#) fueron de gran utilidad, otras no tanto y otras cobraron una utilidad inesperada durante el transcurso del proyecto

### 4.4.1 Valoración de Herramientas Utilizadas

#### 4.4.1.1 Unity

Versátil y veloz, consideramos que Unity cumplió con creces nuestras expectativas.

#### 4.4.1.2 C#

Fue un lenguaje adecuado aunque se perciben algunas limitaciones, en especial con la flexibilidad de tipos de dato a bajo nivel o el control de vida y seguridad de referencias.

#### 4.4.1.3 Mirror

Sorprendentemente veloz y configurable, sentimos que hemos usado muy poco de lo que provee y en ocasiones hemos re-inventado la rueda por no investigar a fondo la documentación de este recurso. Pero fue más que adecuada para nuestra aplicación.

#### 4.4.1.4 MySql, Php, PhpMyAdmin

Esta estrategia para afrontar la necesidad de persistencia de datos fue omitida en favor de AWS Lambda y AWS DynamoDb por su integración nativa con otros servicios de Amazon. Esto tuvo asociado un costo de aprendizaje no menor de todo el entorno de microservicios de Amazon. Este aprendizaje en ocasiones fue frustrante y tedioso pero en definitiva la extensión de documentación disponible, niveles de acceso gratuitos y el aspecto *unmanaged* de los servicios de Amazon fue un gran beneficio para el proyecto.

#### 4.4.1.5 Git y GitHub

Temprano en el desarrollo [descubrimos a Plastic SCM](#), una alternativa para control de versión ofrecida por Unity con integración nativa en sus proyectos. En este caso el juicio de valor puede ser un empate. Si bien la integración con Unity es muy buena a la hora de necesitar combinar archivos binarios propietarios como *prefabs* y *scenes*. Sentimos que hasta el día de hoy estamos aprendiendo como usarla, adicionalmente su restricción de un repositorio por proyecto de Unity generó grandes inconvenientes durante los primeros 6 sprints.

#### 4.4.1.6 Discord, Microsoft Teams, Google Jamboard

Discord resultó invaluable para nuestra comunicación, su integración para varios dispositivos, versatilidad, historial y calidad fueron apreciados. Microsoft Teams fue pasable como herramienta de comunicación con nuestro tutor dejando carencias en algunas funcionalidades básicas como configuración de notificaciones. Google Jamboard fue usado exactamente una vez, en su lugar encontramos [Miro](#) como una herramienta más útil para etapas de tormenta de ideas.

#### 4.4.1.7 Jira, Confluence

Jira, si bien un estándar a nivel industrial para manejo de tareas en proyectos de software probó requerir demasiada atención para ser útil en nuestra escala, combinado con su baja versatilidad en modo Cloud del nivel gratuito es nuestra recomendación para otros equipos de evitar. Una simple planilla de excel puede ser de mejor utilidad.

Confluence por otro lado se mostró mucho mas versátil, aunque requirió extensiones como [PDF Scroll Exporter](#) y considerable configuración inicial, probó a la larga proveer el ideal de colaboración online constante con una producción final de un documento PDF de un solo click.

#### 4.4.1.8 Draw.io

Esta herramienta probó ser invaluable y increíblemente versátil a la hora de visualizar estructuras, diseñar código representar tiempo e incluso organizar tareas.

### 4.4.2 Stack de Tecnologías

El stack de tecnologías cambió desde nuestra concepción original en la etapa de anteproyecto, pero no mucho mas desde nuestra decisión en el Sprint 2 de usar [la infraestructura de AWS](#).

Si vale la pena quizás entrar en detalle de como funcionan las sesiones de juego con GameLift.

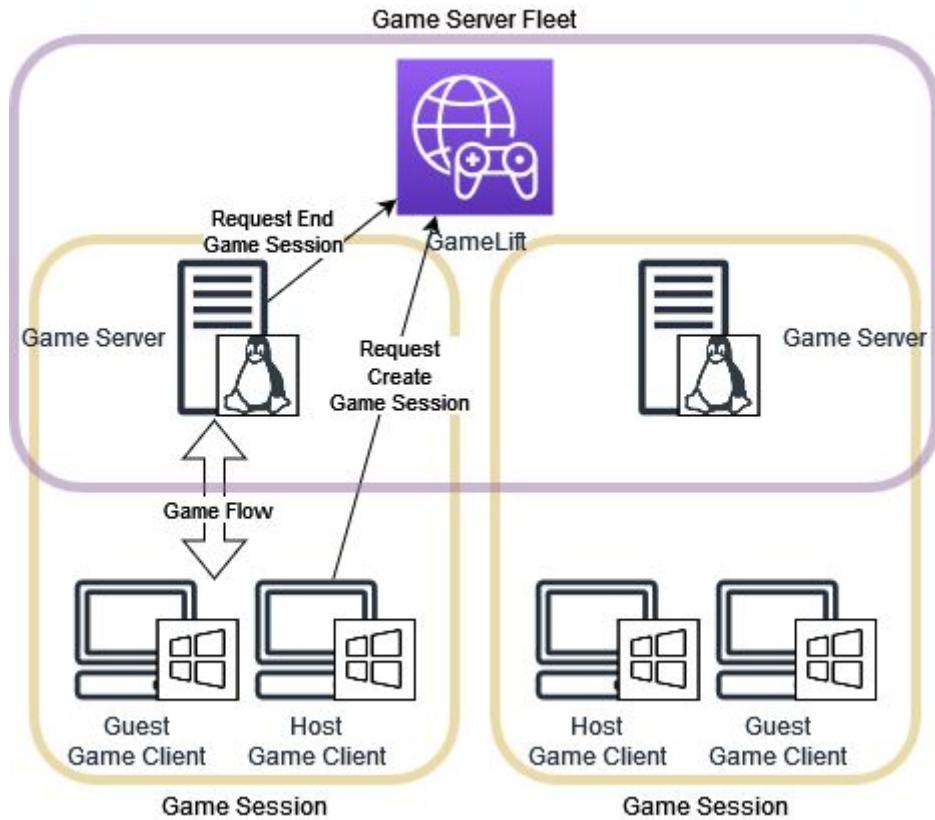


Figura 100. GameLift Fleet

- Una vez subida una *build* de un servidor de Unity compilada en Linux, a GameLift se crea una Fleet.
- Esta Fleet puede virtualizar servidores a partir de nuestro *build* si un cliente autenticado lo requiere.
- El flujo de juego determina lo que sigue, pero el servidor tiene autoridad para requerir la terminación de si mismo concluyendo la sesión de juego, y liberando los recursos.
- Aumentando la cantidad de procesos en nuestra Fleet podemos incrementar la cantidad de sesiones de juego concurrente admitidas.

Usos, beneficios y problemas presentados por las herramientas que planteamos usar y las que usamos finalmente.

## 4.5 Control de Calidad

Con el fin de controlar la calidad del producto, nos adherimos a un estándar de código al cual no todos los integrantes estábamos acostumbrados. Esto significó un leve costo de re-trabajo en algunas ocasiones, pero consideramos que el beneficio en la calidad y

la mejor legibilidad del código compensaron el tiempo invertido. A lo largo del proyecto todos los integrantes nos fuimos adaptando al estándar lo cuál significó una mejora en la productividad.

## 4.6 Método de Trabajo

Inicialmente consideramos Sprints de 2 semanas aproximadamente. Esto probó ser inadecuado ya que muchas de las funcionalidades no podían desarrollarse en ese tiempo. Sin embargo atenernos a este marco de tiempo y usando un software de control de tareas diseñado para otra escala de equipo como Jira, generó la ilusión de avance y la ofuscación de *blockers*.

Estas situaciones combinadas nos dejan con una velocidad de cierre tareas peligrosamente baja. Fue necesario en la última etapa del desarrollo, llevar un control mas *ad-hoc* de las tareas y fue vital elegir fechas límite para grupo de funcionalidades y considerar contingencias cuando las fechas no se alcanzan.

## 4.7 Posibles Mejoras

Identificamos las siguientes mejoras de cosas que nos hubiera gustado desarrollar o mejorar.

### 4.7.1 Mejoras a la Presentación

- Añadir más información contextual del estado actual de las unidades en la partida.
- Añadir más información contextual del estado actual de la partida en general.
- Mejorar el *feedback* de los elementos interactivos de la UI.
- Hacer un repaso de UX para la diagramación de los elementos en pantalla.

### 4.7.2 My Decks

Si bien hemos diseñado el código para admitir mazos personalizables faltó tiempo para poder desarrollar el back end referido a la persistencia de los mazos generados por un usuario y tuvo que ser de-priorizada.

### 4.7.3 Métricas

Sentimos que nuestro proyecto tiene potencial para expandir su valor si tuviéramos una estrategia para capturar e interpretar datos estadísticos del uso del juego por parte de los usuarios, pero nos vimos forzados a dedicar nuestro tiempo a tareas que consideramos de mayor prioridad.

#### 4.7.4 Profundizar la Jugabilidad

Nuestro producto alcanzo una etapa funcional, pero esto no hace a un juego. Es necesario iterar en el game design, el balance, el diseño de las cartas, el ciclo de recompensas para generar un juego con un componente de valor y entretenimiento que honre el tiempo invertido por sus usuarios y desarrolladores. Vemos que tenemos los bloques iniciales para esto pero requiere una inversión de tiempo considerable aún.

### 4.8 Lecciones Aprendidas

#### 4.8.1 Estimaciones

Al momento de estimar horas para tareas, es de tener en cuenta que no todas las horas son iguales, especialmente si los miembros del equipo tienen otras obligaciones durante gran parte del día. 3 horas después de una jornada laboral no son iguales a 3 horas de un día sin otras ocupaciones.

#### 4.8.2 Herramientas

Las herramientas y tecnologías elegidas son importantes, pero más importante aún es un constante cuestionamiento de la utilidad que aportan, una herramienta debería mejorar la productividad siempre, en el momento que no sea así debe ser re-evaluada o reemplazada.

#### 4.8.3 Gestión

La gestión del proyecto no es un asunto menor o algo que se pueda delegar a un software des-centralizado de control de tareas. Los planes iniciales pueden ser incorrectos y continuar adelante en base a esos planes solo porque el software de control de tareas dice que se cierran las tareas que se tienen que cerrar es un error. Es necesario analizar y reanalizar el estado del proyecto de manera frecuente y pragmática.

#### 4.8.4 Colaboración

Pedir ayuda cuando uno está bloqueado es esencial, dos cabezas dedicadas a una tarea pueden avanzar más rápido ya que se aprovechan diferentes puntos de vista para el mismo problema.

## 4.9 Cierre

Agradecemos la oportunidad de haber tenido un marco educativo que nos permitiera desarrollar un proyecto de esta rama de Software, así como la confianza de la institución de permitirnos afrontar un desafío para el cual no íbamos a poder apercibir mucha asistencia académica.

Nos sentimos satisfechos con el contenido producido, teniendo en cuenta limitantes de tiempo y experiencia consideramos que desarrollamos algo de gran valor. Lo aprendido en el transcurso de este proyecto tanto en técnicas de programación, uso de herramientas y trabajo en equipo serán invaluables de aquí en adelante para todos los miembros del equipo.

## 5 Glosario

- [g1] - **Digital collectible card game (DCCG)** - Un juego de cartas colecciónables digital es un videojuego que emula a los juegos de cartas colecciónables (CCG). Los CCG son un tipo de juego de cartas con cartas no predefinidas y de gran variedad, que otorgan individualidad a cada carta, y con las que se construye un mazo.
- [g2] - **Turn-based strategy (TBS)** - Un videojuego de estrategia por turnos, es un videojuego donde el flujo se partitiona en partes definidas y visibles llamadas turnos o rondas. Cuando todos los jugadores han tomado su turno, esa ronda de juego se da por terminada, y comienza la siguiente ronda de juego.
- [g3] - **2.5D** - Término que se utilizan para describir proyecciones gráficas en 2D y técnicas que hacen que una serie de imágenes o escenas parezcan ser de tres dimensiones (3D), cuando en realidad no lo son. Alternativamente, puede describir a un videojuego tridimensional que se limita a un plano de dos dimensiones.
- [g4] - **Milestone build** - Versión del software que marca una etapa significativa del desarrollo.
- [g5] - **Motor de desarrollo de videojuegos** - Serie de librerías de programación que asisten en el diseño y el desarrollo de videojuegos.
- [g6] - **Feedback** - Información evaluativa/correctiva sobre una acción, evento o proceso que se dirige a la propia fuente de la acción, evento o proceso.
- [g7] - **Input** - Información alimentada a un sistema de procesamiento de datos, computadora o similar.
- [g8] - **Lobby** - En videojuegos, un *lobby* es una zona de espera virtual donde los jugadores aguardan hasta que las condiciones para iniciar un juego online se cumplan.
- [g9] - **Prefab (Unity)** - Los *prefabs* son un tipo especial de componente que permite guardar en el proyecto *GameObjects* totalmente configurables para su reutilización. Éstos pueden ser compartidos por distintas escenas, y hasta por distintos proyectos.

# 6 Bibliografía

Una bibliografía

## 6.1 Vínculos Externos

- [v1] - <https://playhearthstone.com/en-us>
- [v2] - <https://magic.wizards.com/en>
- [v3] - [https://en.wikipedia.org/wiki/Pok%C3%A9mon\\_Trading\\_Card\\_Game](https://en.wikipedia.org/wiki/Pok%C3%A9mon_Trading_Card_Game)
- [v4] - [https://en.wikipedia.org/wiki/Yu-Gi-Oh!\\_Duel\\_Monsters](https://en.wikipedia.org/wiki/Yu-Gi-Oh!_Duel_Monsters)
- [v5] - <https://en.wikipedia.org/wiki/Chess>
- [v6] - <https://en.wikipedia.org/wiki/Checkers>
- [v7] - [https://en.wikipedia.org/wiki/Civilization\\_\(series\)](https://en.wikipedia.org/wiki/Civilization_(series))
- [v8] - [https://en.wikipedia.org/wiki/XCOM\\_2](https://en.wikipedia.org/wiki/XCOM_2)
- [v9] - <http://www.arfgamestudio.com/>
- [v10] - [realvirtualitygames@gmail.com](mailto:realvirtualitygames@gmail.com)
- [v11] - <https://unity.com/>
- [v12] - <https://www.linkedin.com/in/manuel-atienza-lombardo>
- [v13] - [https://docs.aws.amazon.com/es\\_es/lambda/latest/dg/welcome.html](https://docs.aws.amazon.com/es_es/lambda/latest/dg/welcome.html)
- [v14] - <https://aws.amazon.com/dynamodb/>
- [v15] - <https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/dotnetv3/dynamodb#code-examples>
- [v16] - <https://www.youtube.com/watch?v=X4ZkZEfVFJE>
- [v17] - <https://docs.unity3d.com/Manual/class-ScriptableObject.html>
- [v18] - <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/>
- [v19] - <https://marketplace.visualstudio.com/items?itemName=AmazonWebServices.AWSToolkitforVisualStudio2022>
- [v20] - <https://github.com/pschraut/UnityCommandLine>
- [v21] - <https://docs.unity3d.com/Manual/AnimationOverview.html>
- [v22] - <https://docs.unity3d.com/Manual/class-AnimatorController.html>
- [v23] - <https://docs.unity3d.com/Manual/AnimatorOverrideController.html>
- [v24] - <https://assetstore.unity.com/packages/2d/environments/galaxy-armada-pixel-art-spaceships-fleet-vol1-68461>

## 6.2 Referencias Bibliográficas

- [1] - Sommerville, Ian (2005) Ingeniería de Software, 7a ed. Madrid, España: PEARSON EDUCACIÓN. S.A. ISBN: 84-7829-074-5

[2] - Knuth, Donald Erwin (1981) Seminumerical algorithms. The Art of Computer Programming, 2a ed. Reading, MA, Estados Unidos: Addison–Wesley.  
ISBN 0-201-03822-6.

## 7 Anexos

### 7.1 Manual de Usuario

#### 7.1.1 Hoodies

Hoodies es un juego de estrategia en turnos donde los jugadores poseen mazos de cartas que usan para invocar naves, con el objetivo de destruir la nave nodriza enemiga.

##### 7.1.1.1 Primeros pasos

Los jugadores deben acceder al repositorio de hoodies en Git <https://github.com/SpaceGauchoDev/Hoodies> y descargar el archivo `FinalDistribution.zip`.

El *readme* del repositorio tiene explicitado todos los pasos necesarios para comenzar a jugar Hoodies.

##### 7.1.1.2 Creación de una Partida

Para comenzar una partida los jugadores deben dirigirse a “Private Match”. Una vez dentro de este menú, un jugador tiene que ser el *host*, y el otro debe unirse a esa partida. En caso de tratarse de una partida *on-line*, el jugador que se une a la partida tiene que ingresar el código proveído por el *host*. Esto se explicará en más detalle más abajo.

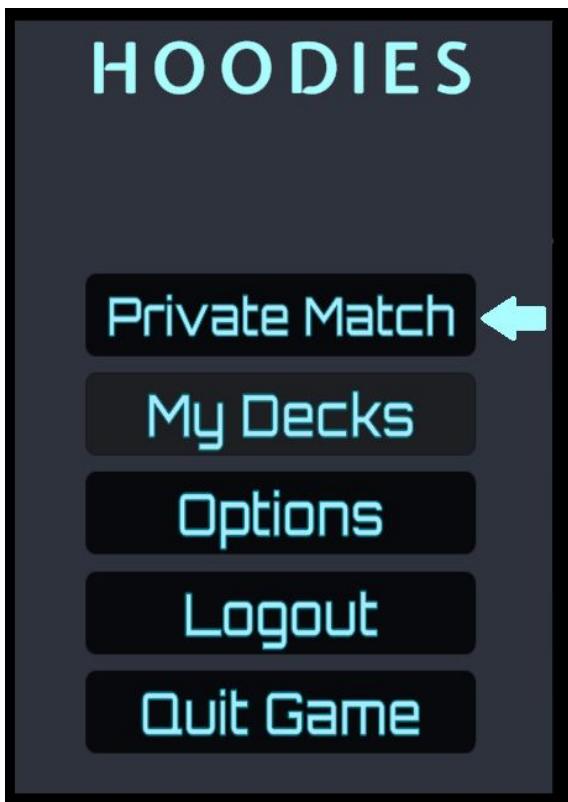


Figura 101. Menú principal

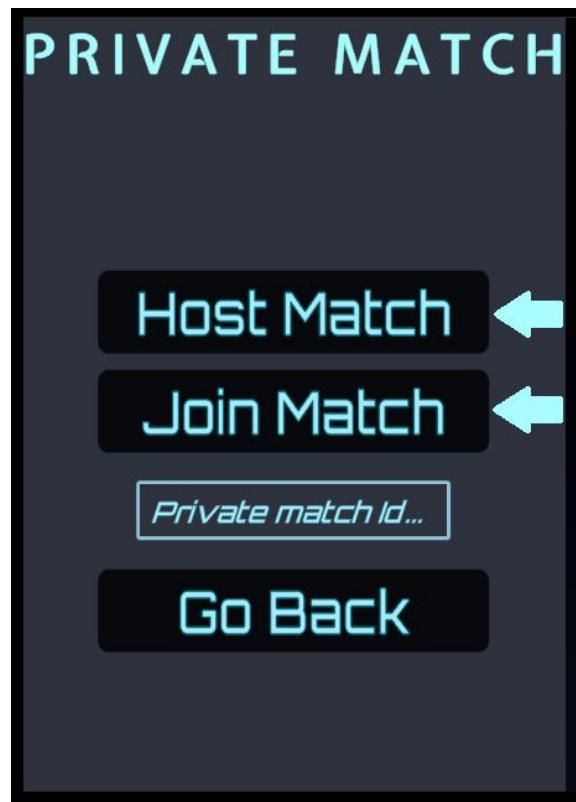


Figura 102. Menú de Private Match

### 7.1.1.3 Lobby

Una vez que el *host* seleccionó el botón “Host Match” —si se pudo establecer una conexión con el servidor— se muestra la siguiente pantalla.



Figura 103. Estado inicial de un lobby desde el punto de vista de un host.

En la parte izquierda se puede ver el *private match ID* y el panel de usuario del jugador local. Como en este momento sólo hay un jugador en el lobby, el *host* debe comunicar el *private match ID* (subrayado en rojo) al otro jugador.

Ingresando este código en el campo “Private match Id...”, el segundo jugador presiona el botón “Join Match” y entra al *lobby*.



Figura 104. Pantalla lobby cuando dos jugadores se han unido.

Cuando ambos jugadores se encuentren dentro del lobby, el botón “Ready” se habilita, y una vez que ambos seleccionaron este botón, comienza un contador de 15 segundos. Una vez que el contador llega a los 5 segundos, ambos jugadores pierden la opción de irse de lobby. Cuando se llega a los 0 segundos, el juego comienza.

#### 7.1.1.4 Match

En un *match*, los jugadores ven la siguiente pantalla.

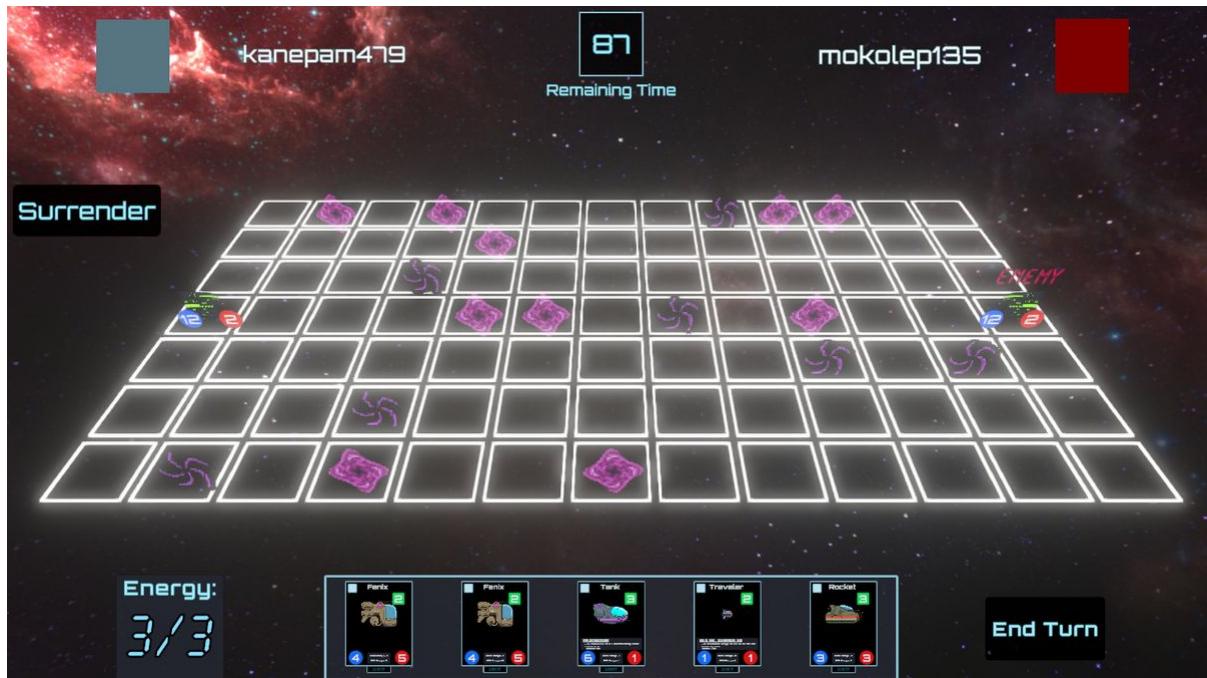


Figura 105. Pantalla de Match al inicio de una partida

Elementos en pantalla al comienzo de la partida

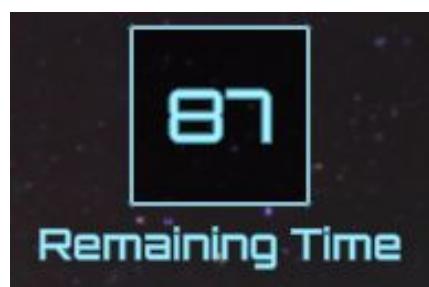


Figura 106. Contador

Indica cuánto tiempo tiene el jugador actual para hacer sus acciones antes de que su turno se vea forzosamente terminado.



Figura 107. Nombres de los usuarios

Muestra los nombres de los jugadores. Corresponde con el lado del tablero en el que comenzaron la partida.



Figura 108. Energía

Ciertas acciones en el juego cuestan energía. Este elemento muestra cuánta energía el jugador tiene disponible del lado izquierdo de la fracción, y cuánto es su máximo de energía en ese turno. En cada turno propio, este máximo se incrementa por 1 hasta un máximo de 10 y toda la energía gastada en el turno anterior se recupera.



Figura 109. Mano

Esta es la mano del jugador, aquí el jugador puede ver qué cartas tiene disponibles en ese turno. Al comienzo de cada turno propio, el jugador recibe una nueva carta de su mazo siempre y cuando su mano tenga menos de 5 cartas.



Figura 110. Botón “End Turn”

Con este botón el jugador puede terminar su turno antes de que el contador llegue a su mínimo.

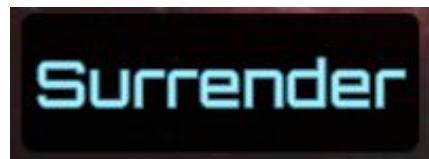


Figura 111. Botón “Surrender”

Con este botón el jugador puede rendirse y terminar la partida.

## Gameplay

### Naves nodrizas

Al comenzar la partida, ambos jugadores comienzan con una nave nodriza (*mothership*) de su lado del tablero. El objetivo de cada usuario es destruir esta unidad. Si un usuario pierde su nave nodriza, pierde el juego.



Figura 112. Carta de mothership



Figura 113. Sprite de mothership

### Unidad

Para cumplir con el objetivo de destruir la nave nodriza contraria, cada jugador podrá invocar distintas unidades de su mano, arrastrándolas de su mano al tablero. Una vez que se comienza a arrastrar una carta, el jugador verá que ciertas celdas se iluminarán de color azul. Esta es la zona donde se puede invocar una nueva carta.

Esta zona es determinada por la nave nodriza, que dibuja un cuadrado a su alrededor, y las unidades *spawner*, que habilitan a invocar en sus celdas adyacentes.

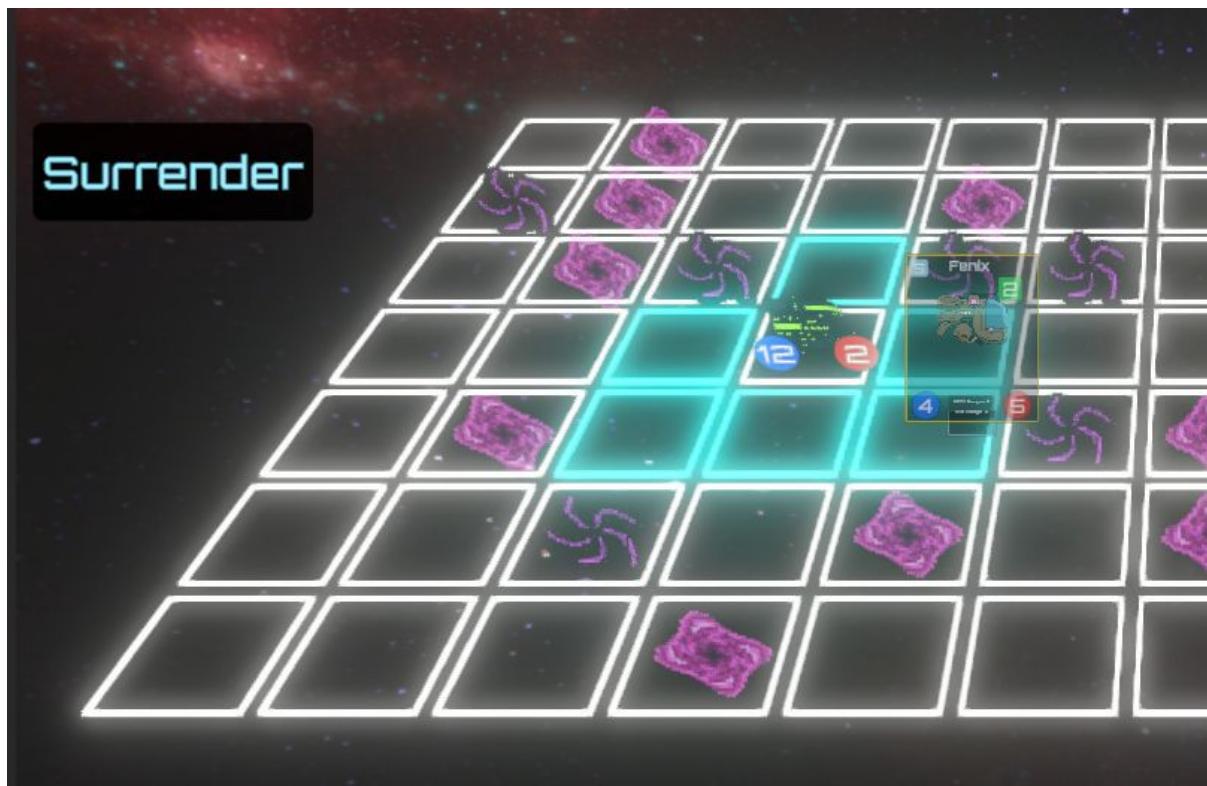


Figura 114. Área de invocación valida con sólo la nave nodriza.

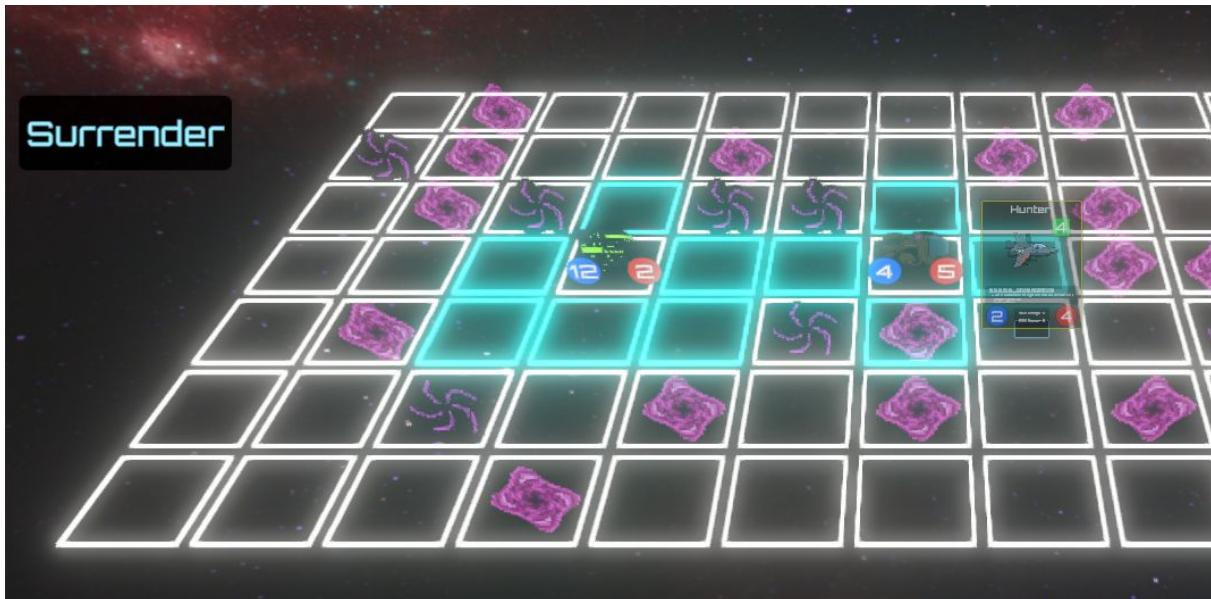


Figura 115. Área de invocación expandida por una unidad spawner.

### Unidad

Las cartas existen en el mazo del jugador, y en la partida se pueden ver en la mano. Cada carta contiene información sobre la unidad que invocan.

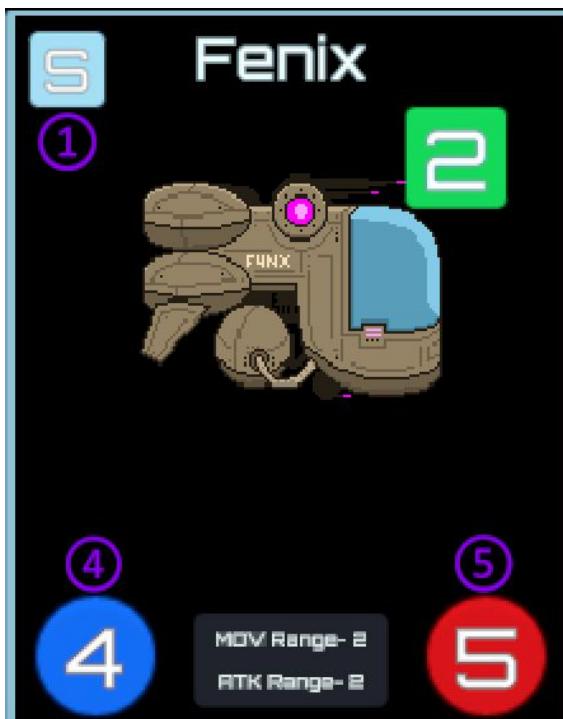


Figura 116. Ejemplo: Fenix

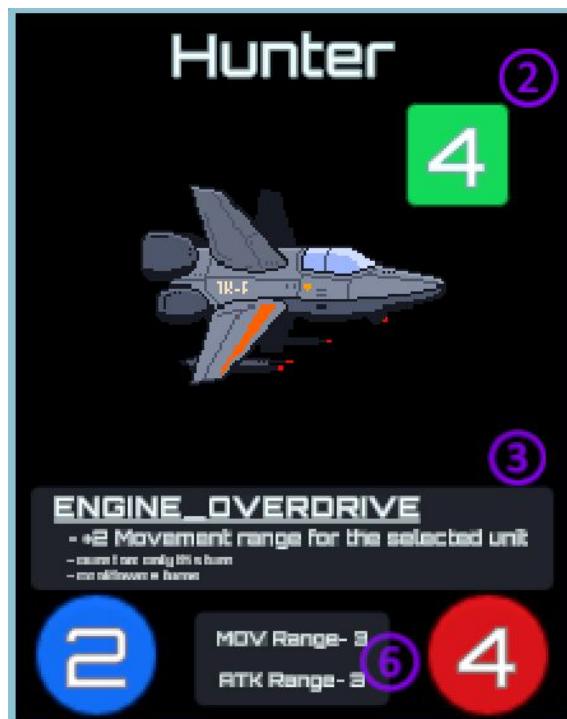


Figura 117. Ejemplo: Hunter

1. Esta 'S' muestra si esta unidad es *spawner*. Una unidad *spawner* habilita a invocar a otras unidades en sus celdas adyacentes.
2. Este número indica el costo de invocar esta carta. Si el usuario tiene menos energía disponible que el costo de la unidad, no podrá ser invocada.
3. En este lugar se describa la habilidad que tiene una unidad, en caso de tener una. También se indica qué hace esa habilidad en particular. La duración (*duration*) indica cuántos turnos esa habilidad durará en el tablero. El *cooldown* indica cuántos turnos tiene que esperar esta unidad para activar su habilidad nuevamente una vez activada.
4. Este número indica el escudo de la unidad. Si llegan a 0, la unidad es destruida.
5. Este número indica los puntos de ataque de una unidad.
6. En este lugar se muestran el rango de movimiento de una unidad, y su rango de ataque, respectivamente.

### Control de las Unidades

Una vez que un jugador hace click sobre una unidad en su control, las celdas a donde puede moverse se iluminarán de amarillo y en caso de tener una unidad hostil en rango de ataque, la celda de esta última se iluminará de rojo.

Si esta unidad tiene una habilidad, un botón aparecerá a la derecha de la pantalla. Las habilidades se explicarán más abajo.

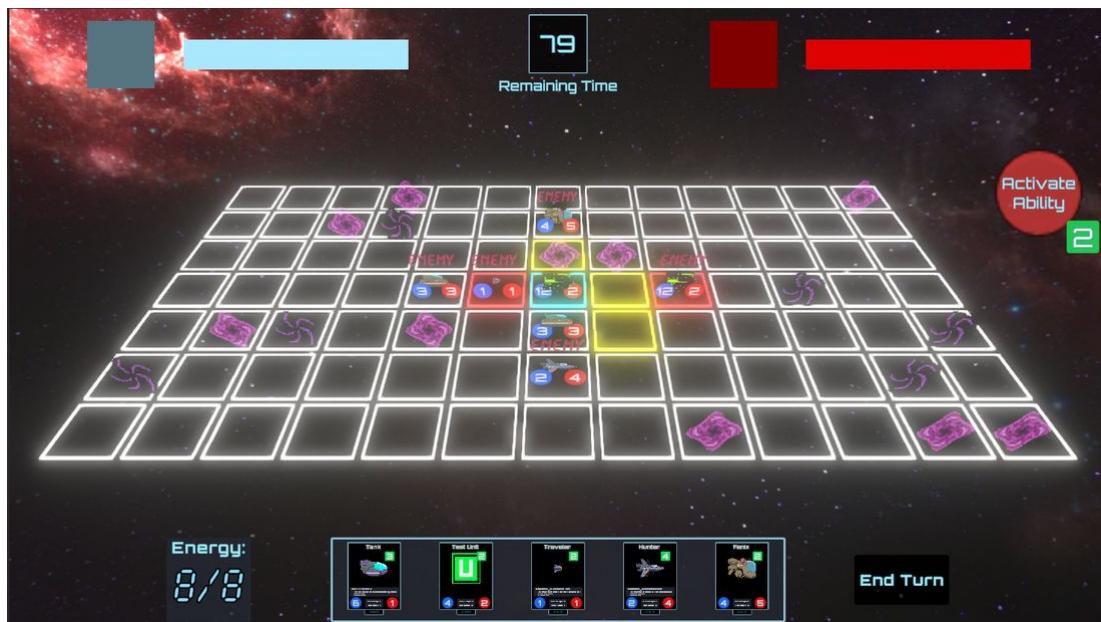


Figura 118. Posibles movimientos y posibles ataques de esta unidad

El jugador entonces puede hacer click derecho en cualquiera de estas celdas amarillas y apretar el botón que aparece en pantalla para moverse. Cada unidad sólo puede moverse un máximo de una vez por turno.

Antes o después de moverse, el jugador puede optar por atacar, si esta acción es posible, o activar su habilidad, si el costo indicado cerca del botón lo permite.

Sea cual sea su elección, una vez atacado o activado su habilidad, esa unidad no podrá ser usada por el resto del turno.

Si el jugador mantiene su mouse encima de la celda donde se encuentra una unidad, el jugador podrá consultar todos los datos base de esa unidad.

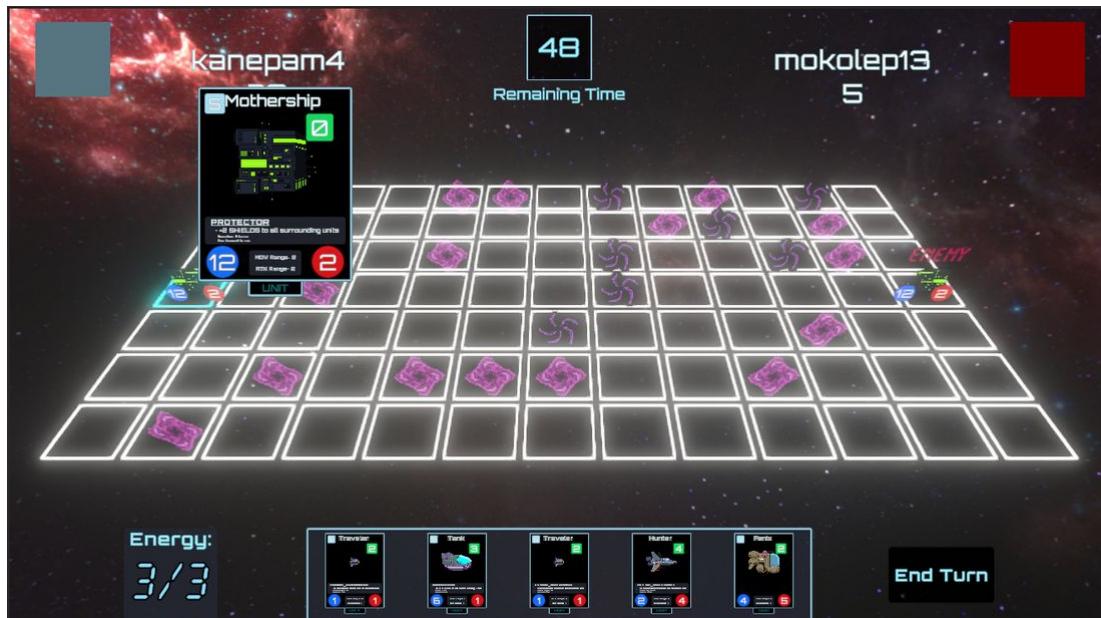


Figura 119. Tooltip de una carta

## Combate

El combate entre dos unidades es mutuo, es decir, ambas unidades se van a ver dañadas por el equivalente del ataque de la unidad contraria.

Una unidad sólo puede atacar a otra que esté en su rango de ataque y con las celdas entre medio desbloqueadas. Las celdas pueden bloquearse con la presencia de otras unidades, o por celdas especiales como el agujero negro.

## Habilidades

Cuando se presiona el botón para activar una habilidad, el jugador verá que la zona en que es posible activar esta habilidad se iluminará en el tablero. El jugador debe llevar el mouse a una de estas celdas y la habilidad se activará en la zona indicada. Si el jugador está tratando de activar la habilidad en una celda inválida, esta zona se marcará en rojo.

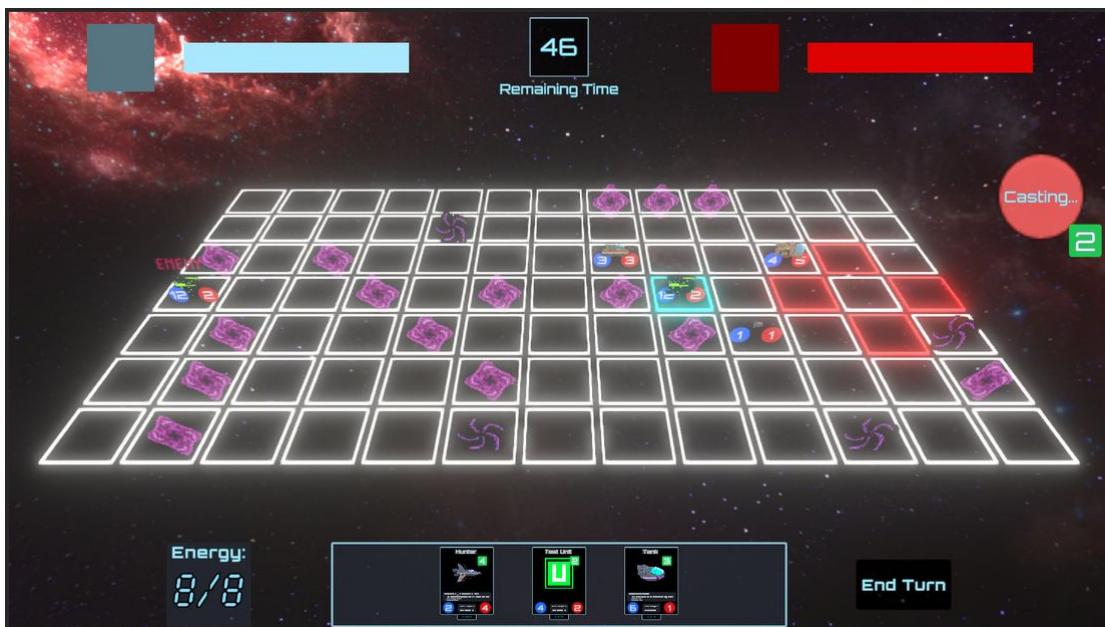


Figura 120. Área inválida

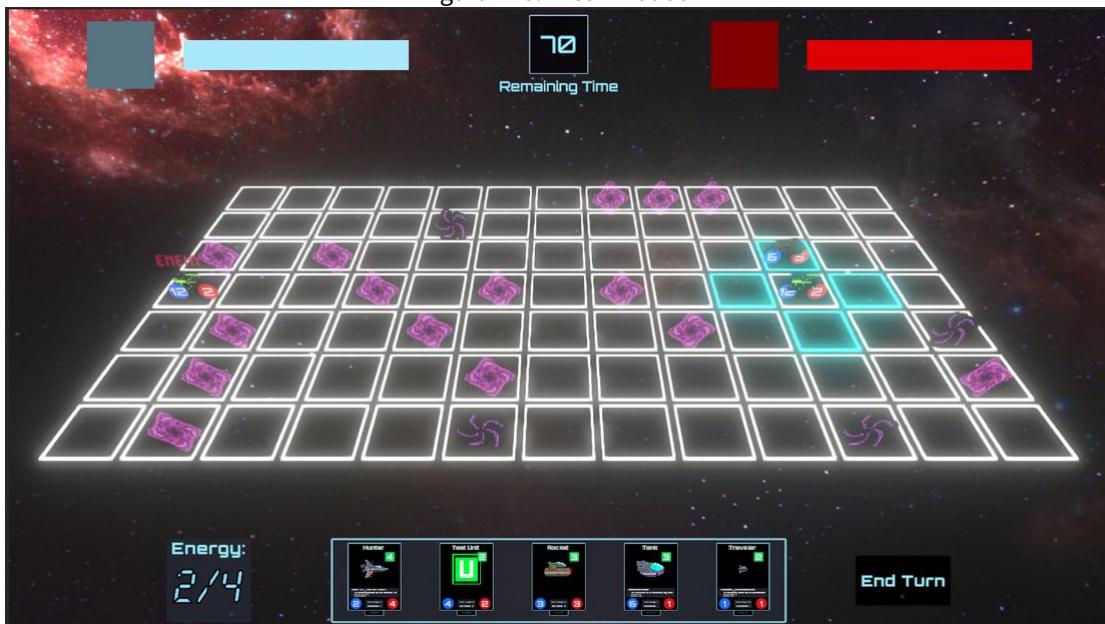


Figura 121. Área válida

Cada tipo de habilidad es único, por lo que el jugador debe leer la descripción de cada habilidad en la carta de su unidad para conocer su efecto.

Como fue explicado anteriormente, las habilidades tienen duración, que indica cuántos turnos la habilidad durará en el tablero, y *cooldown*, que indica cuántos turnos tiene que esperar la unidad para activar su habilidad nuevamente una vez activada.

En este momento, existen dos habilidades en el juego:

- Protector: “Protector” crea un aura en las celdas adyacentes a la unidad que lo activó que agregan 2 a los escudos de cualquier unidad que se encuentre en ellas. Este efecto sólo se aplica si las unidades afectadas se mantienen en esta zona. La habilidad dura 4 turnos y se puede activar cada 8.
- Engine Overdrive: “Engine Overdrive” agrega 2 al rango de movimiento a la unidad que lo activó. El efecto dura 4 turnos y se puede activar cada 6

## Celdas

En el tablero existen tres tipos de celdas distintas: Vacía, Nébula y Agujero Negro.

- Vacía: El tipo de celda más común. No tiene ninguna propiedad especial en particular.
- Nébula: Las nébulas dificultan el recorrido de una unidad en el mapa, costando el doble de esfuerzo que una celda vacía. A su vez, afectan los ataques, restándole 1 al rango de ataque posible de una unidad en si se encuentra en el camino de un ataque.
- Agujero Negro: Esta celda es intransitable y bloquea todos los ataques si se encuentra en el camino de un ataque.

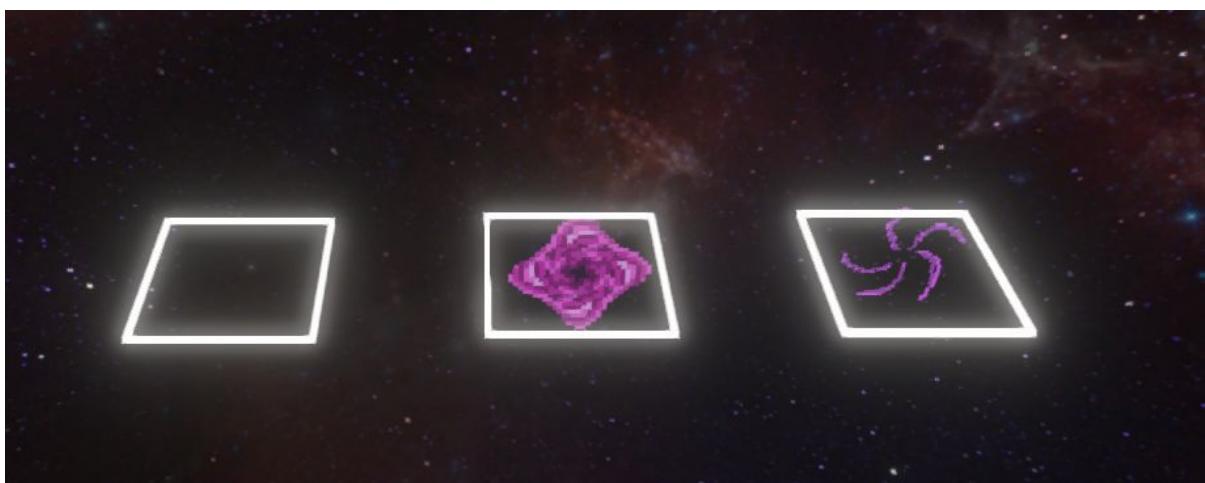


Figura 122. De izquierda a derecha: Celda vacía, celda nébula, y celda agujero negro.