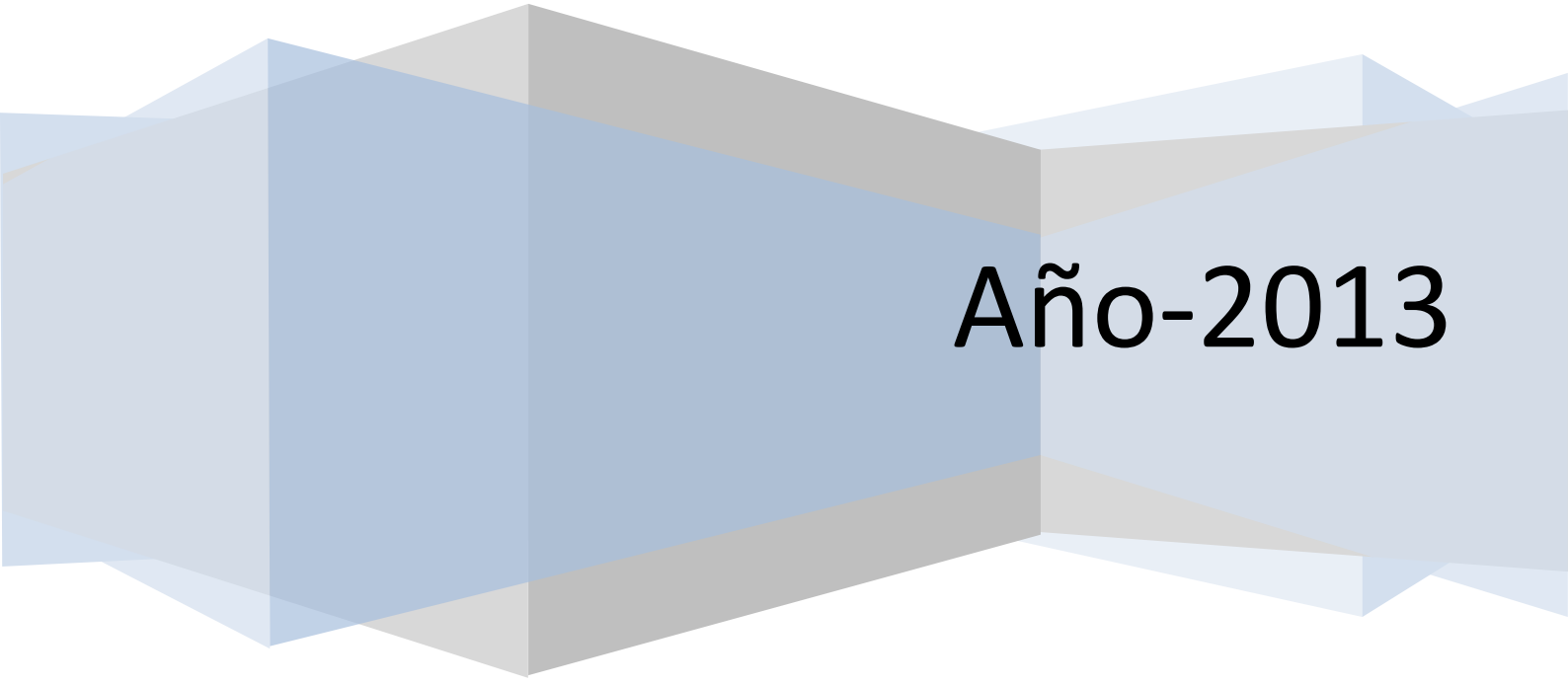
P.U Pets

Planificación del proyecto.

Jesús Miguel García Canales



Año-2013

Dedicado a mi madre, por apoyarme a pesar de mis inquietudes poco comunes.

A mi padre, por transmitirme fuerza en los momentos delicados.

A mi hermano, por estar conmigo a pesar de todo.

A Jose, por esos debates interminables y por ayudarme siempre a planear el siguiente paso.

A Ramon, por ayudarme siempre que estaba en su mano y por reforzarme en mis pensamientos.

A Javi que además de ser el grafista con más talento que conozco, comparte mi camino y mis metas.

Y sobre todo, se lo dedico a Natalia, mi Nat. La persona que me guió cuando más perdido estaba, que me apoyó cuando estaba cayendo y que sufre conmigo en mis malos momentos. Sin ti, nada de esto hubiera sido posible. Sin ti, seguiría pensando que dedicarse a esto es una utopía. Espero poder devolverte algún día todo lo que me aportas.

Índice

1 PLANIFICACIÓN

1.1 INTRODUCCIÓN

En el presente documento, se desglosa la realización del proyecto P.U Pets en tareas y se organiza la elaboración de estas conforme al calendario y al tiempo disponible.

1.2 DESCRIPCIÓN

El proyecto P.U Pets, se trata de un videojuego que será desarrollado para la plataforma Android como TFC para la Universidad Oberta de Catalunya. El juego constará de 5 fases/niveles completos.

1.3 PLANIFICACIÓN

1.3.1 ANÁLISIS PREVIO

Objetivo:
Durante esta fase quedará definido tanto el tipo de juego que queremos desarrollar como su género. Para ello habrá que tener en cuenta diversos factores:

- Software utilizado.
- Público Objetivo.
- Posible competencia.

Documentación:
Se documentará el género escogido poniendo ejemplos de proyectos anteriores de similares características, además de justificar cada elección hecha.

Finalidad y riesgos:
La finalidad de esta fase es servir como punto de partida. Solo cuando se ha tomado esta decisión, se puede pasar a la fase de "Diseño de Gameplay", puesto que proceder al diseño sin tener una idea clara de que es lo que se pretende obtener, puede llevar a que el desarrollo se dilate en el tiempo.

1.3.2 DISEÑO GAMEPLAY

Objetivo:
En esta fase se realiza un esbozo inicial de las bases jugables del proyecto.

Documentación:
La documentación en esta fase es especialmente crítica. Supondrá el pilar fundamental del desarrollo, puesto que en ella quedará reflejada las bases fundamentales del proyecto final. Se documentaran minuciosamente los siguientes aspectos:

- Mecánica jugable.
- Historia, temática, ambientación.
- Diseño de GUI (Interfaz Gráfica de Usuario).

Finalidad y riesgos:
La finalidad es conseguir una mecánica simple y divertida aunque puede ocurrir que durante la fase de prototipado detectemos deficiencias en diseño. Tendremos entonces que retomar esta parte con el propósito de corregir insuficiencias.

1.3.3 ANÁLISIS TÉCNICO

Objetivo:
Análisis en el que definimos las distintas partes que necesitaremos implementar en nuestro framework, teniendo en cuenta todos sus aspectos fundamentales.

Documentación:
Será imprescindible documentar los siguientes puntos:

- Periféricos que usará el usuario para interactuar con el videojuego (Pantalla táctil, teclado etc.).
- Versiones de Android compatibles.
- Permisos necesarios en archivo declarativo XML.

Finalidad y riesgos:
El objetivo de esta fase es definir los aspectos más técnicos del desarrollo. Estas decisiones son cruciales, puesto que el tener que hacer cambios en esta parte una vez comenzada la implementación de la parte jugable, puede resultar muy complicado y provocar errores o prolongaciones en el tiempo necesario para la codificación.

1.3.4 DISEÑO TÉCNICO

Objetivo:
Elaboración de diagramas técnicos necesarios para la fase de desarrollo.

Documentación:
Será imprescindible realizar los siguientes diagramas:

- Diagrama de clases.
- Diagramas de estados.

Finalidad y riesgos:
Estos diagramas son fundamentales para la fase de implementación. Una deficiencia en el diseño técnico puede acarrear múltiples problemas en esta fase, por lo que es recomendable poner máxima atención. Además, la decisión de para que versiones de Android es compatible nuestros proyecto, puede definir el público objetivo, ya que el número de móviles de última generación siempre será más reducido que la gama media o baja.

1.3.5 IMPLEMENTACIÓN FRAMEWORK

Objetivo:
Programación completa tanto del framework como del archivo declarativo XML que usaremos para implementación de la mecánica jugable.

Documentación:
Será necesario documentar los siguientes aspectos:

- Permisos necesarios derivados del archivo declarativo.
- Clases con las que se han implementado los dispositivos.

Finalidad y riesgos:
La interacción programa-usuario es la base de todo videojuego. Al terminar esta fase, esa interacción debe ser perfectamente funcional.

1.3.6 PROTOTIPADO

Objetivo:
Al plasmar la idea sobre el papel, es posible que hayamos pasado por algo algún detalle que puede empobrecer la experiencia o hacer menos divertido el proyecto. Por eso, necesitamos un prototipo de la base del gameplay, poniendo especial atención en la GUI.

Documentación:
En esta fase, solo documentaremos en que parte hemos centrado nuestro prototipo.

Finalidad y riesgos:
La finalidad del prototipo es asegurar un diseño jugable solido, divertido y sin carencias.

1.3 7 PRUEBAS

Objetivo:
El objetivo de esta fase es, simplemente, poner a prueba la robustez del diseño de nuestro gameplay a través del prototipo desarrollado en la fase anterior.

Documentación:
Se deberá documentar cada deficiencia detectado en el prototipo al someterlo a prueba y la solución escogida para esta.

Finalidad y riesgos:
Conseguir un diseño óptimo es, probablemente, la diferencia entre un gran videojuego y uno mediocre. Poner a prueba nuestra idea, es por lo tanto, completamente necesario si queremos conseguir calidad.

1.3.8 DESARROLLO

Objetivo:

Una vez probada la mecánica jugable, implementaremos el juego completo. Será necesario programar las diversas fases o niveles. El desarrollo del proyecto, se dividirá en las siguientes sub-tareas:

- Implementación distintas clases jugables.
- Implementación IA.
- Implementación Fase 1.
- Implementación Fase 2.
- Implementación Fase 3.
- Implementación Fase 4.
- Implementación Fase 5.

Documentación:

Será necesaria documentación para cada fase o nivel implementado, así como la especificación de la IA y de las distintas clases jugables.

Finalidad y riesgos:

Durante este desarrollo es posible cometer errores que no se perciben inmediatamente, comúnmente denominados “glitches” o algunos que pueden resultar fatales, denominados “bugs”. Será necesario volver a esta fase, y depurar los errores detectados durante el testing.

1.3.9 TESTING

Objetivo:

Hacer una búsqueda minuciosa de errores. Se dividirá en el siguientes sub-tareas:

- Localización Errores.
- Elaboración de informe de errores.
- Corrección de los errores localizados.

Documentación:

Se deberá elaborar un informe de errores, para poder corregirlos correctamente.

Finalidad y riesgos:

La finalidad de esta fase es localizar todos los errores para poder corregirlos todos y obtener un proyecto final sólido. Habrá que hacer una estimación del tiempo necesario para la localización de los errores y su posterior resolución.

1.3.10 DOCUMENTACIÓN

Objetivo:

La elaboración de la documentación final del proyecto.

Documentación:
Serán necesarios los siguientes documentos:

- Memoria final del proyecto
- Manual de usuario.

1.4. DIAGRAMA DE GANTT

FIGURA 1:

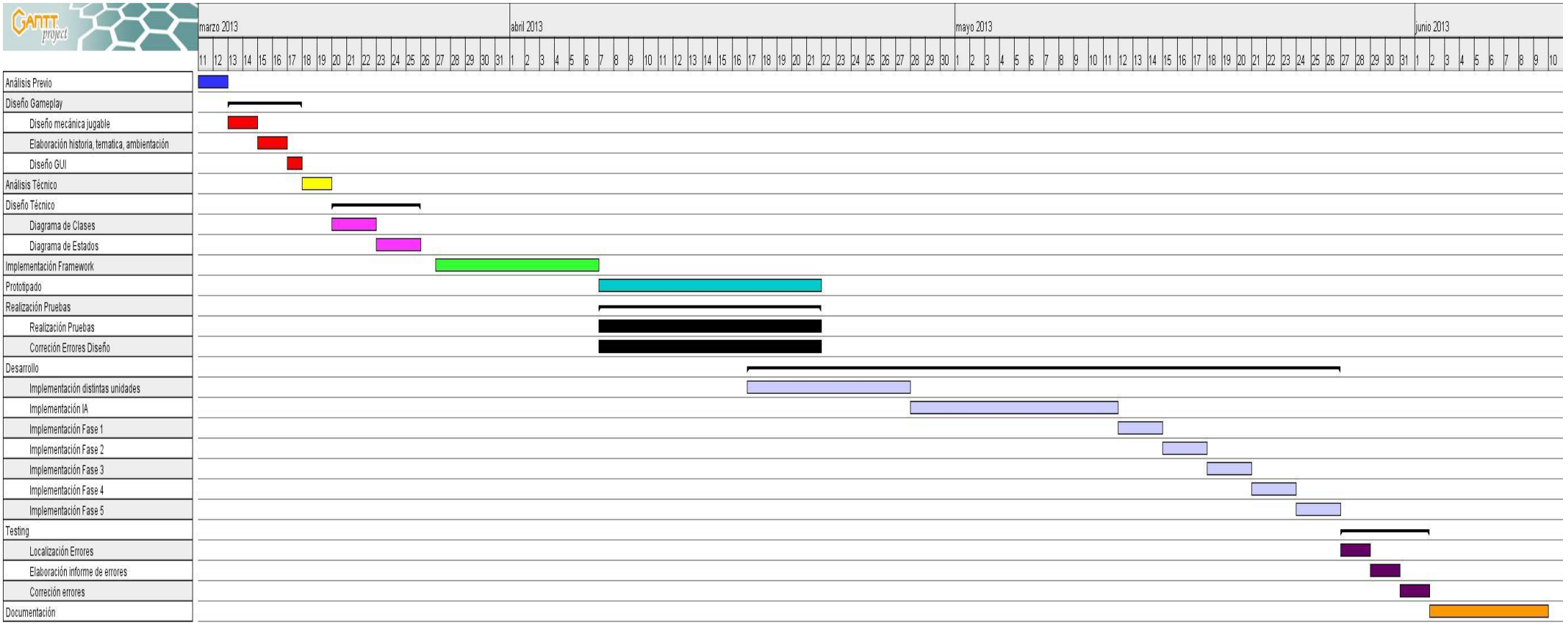


DIAGRAMA DE GANTT DEL PROYECTO: Las sub-tareas de "Realización Pruebas" se realizarán simultáneamente, por lo que, en el caso de encontrar un error de diseño durante al pruebas realizadas al prototipo, se pasará directamente a su corrección.

2 ANÁLISIS PREVIO

En este apartado, sentaremos la bases del proyecto P.U Pets. Para ello deberemos justificar cada decisión tomada en puntos de especial relevancia.

2.1 Software utilizado

P.U Pets será desarrollado con la plataforma Android como objetivo. Al ser un S.O de código abierto, así como la gran cantidad de documentación disponible, proporcionada por la propia Google, además de la enorme comunidad de desarrolladores que suponen un soporte de incuestionable valor, hace que su valor didáctico sea incuestionable. Todo ello convierte Android en la plataforma perfecta para comenzar un proyecto como este.

El entorno de programación utilizado será "adt bundle for Windows" de 32 bits. Se trata del SDK de Android y nos proporciona lo siguiente:

- Eclipse + ADT plugin.
- Herramientas Android SDK.
- Herramientas de la plataforma Android.
- Últimas versiones de la plataforma Android.
- Un emulador funcional de Android.

Todas estas herramientas son software libre, por lo que nos ahorramos pago de licencias de uso, además de un gran soporte técnico a través de las anteriormente mencionadas comunidades de desarrolladores.

2.2 Público Objetivo

Trabajando en la industria del videojuego, nuestro objetivo siempre debe ser llegar al mayor número de personas posible.

El gran público de los videojuegos Android se encuentra entre los llamados" jugadores ocasiones". El poder incluir videojuegos en un elemento de gran necesidad como son los teléfonos móviles, ha permitido que personas que no había comprado una videoconsola nunca, se acercasen a esta industria y comenzasen a consumir este tipo de entretenimiento.

Sin embargo, no hay que olvidar a los jugadores habituales. Aunque son menor en número, su gasto en videojuegos es mucho mayor. Además en este target de mercado es donde se alojan las grandes "comunidades fan", que adquiere todo tipo de merchandising o productos relacionados (como música, ropa o juguetes) con el juego o saga que admiran.

Para ello deberemos ofrecer un juego asequible, capaz de ser jugado por los jugadores menos habituales, pero con suficientes alicientes como para ser atractivos para los jugadores más tradicionales.

Por este motivo, se ha considerado en género de estrategia por turnos como el optimo. Su base jugable es muy simple (como un juego de tablero), pero tiene suficientes alicientes como para resultar atractivo a todos los públicos.

2.3 Juegos Similares y Competencia

El género escogido para el proyecto, es estrategia por turnos, cuyos máximos exponentes son las sagas *Advanced Wars* (Nintendo) y *Panzer General* (Strategic Simulation Inc.). Es un género con un gran número de seguidores y debido a su filosofía y a su gameplay, se ajusta perfectamente a las características de los dispositivos móviles. Su realización no supondría una excesiva carga de trabajo, ni en su apartado técnico (todos los personajes se mueven a través de una cuadricula, por lo que las rutinas no son especialmente complicadas), ni en su apartado artístico (no se precisan de animaciones complicadas).

Ilustración 2: Advanced Wars. Nintendo

Ilustración 3: Panzer General: SS Inc.

Otra ventaja, es que no existen demasiados juegos similares en el mercado móvil, lo que siempre supone un gran atractivo a la hora de vender un producto y es un valor añadido. Los juegos de estrategia disponibles en el Play Market son en su mayoría del tipo “Defend Your Catle”, que es un genero tremendamente popular pero un tanto sobre-explotado. Uno de los juegos más similares en planteamiento a nuestro proyecto P.U Pets, es “King's Bounty: Legions”, sin embargo se trata en realidad de un Strategic Role Playing Game, más enfocado a jugadores tradicionales.

Ilustración 4: King´s Bounty.

Por lo tanto, nuestro proyecto no tendría mucha competencia en Android, algo muy complicado si tenemos en cuenta la cantidad de videojuegos disponibles para este mercado.

3 DISEÑO GAMEPLAY

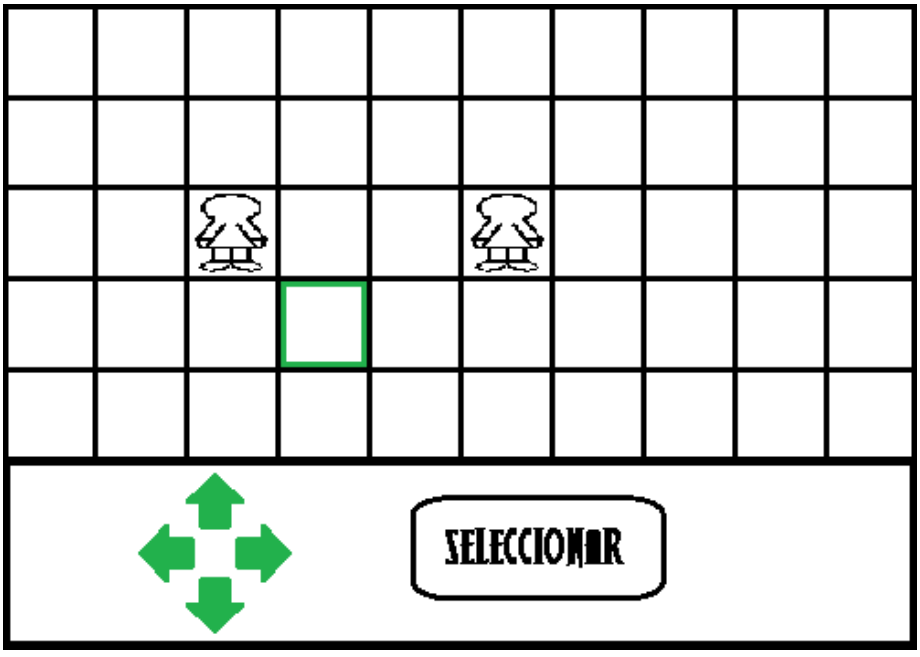
El pilar fundamental de todo videojuego es el gameplay o mecánica jugable. El objetivo de esta parte del proyecto es conseguir unas bases jugables bien equilibradas, lo suficientemente simples como para resultar divertidas de inmediato y con suficiente complejidad como para no resultar monótona. Deberemos definir que características diferenciadores tendrá P.U Pets, con respecto del resto del género de la estrategia por turno.

3.1 Mecánica jugable

En "P.U Pets", hay dos facciones enfrentadas que lucharan distribuyendo sus tropas por todo el campo de batalla o tablero. Ambas facciones podrán fabricar más tropas con el consumo de varios marcadores, como la radioactividad, el carbón, o la energía eólica. El repertorio de tropas que las facciones pueden fabricar es el mismo, y las tropas solo variarán en el aspecto con respecto a las del rival.

Las estrategias (tanto del jugador como de la IA) se llevan a cabo por turnos alternos. El jugador mueve un indicador a través de la cuadricula en la que se divide el campo de batalla por medio de la GUI dispuesta en la pantalla táctil.

Ilustración 5: Ejemplo de tablero de juego.



Puede seleccionar cualquier unidad de su facción para moverla, atacar, usar sus movimientos especiales o conquistar alguna central eléctrica. También puede seleccionar unidades enemigas con fines informativos, como comprobar cuántos HP (Health Points) le quedan o su estado.

Una vez seleccionada una de nuestras unidades, lo primero que se presentará será el rango de movimiento de dicha unidad, en forma de cuadros resaltados. Estos cuadros son por los cuales se puede mover. La amplitud de este rango dependerá de varios factores como:

- **Especie de la unidad:**

  Como hemos explicado anteriormente, las distintas especies tienen distintas características, una de ellas será el rango de movimiento.

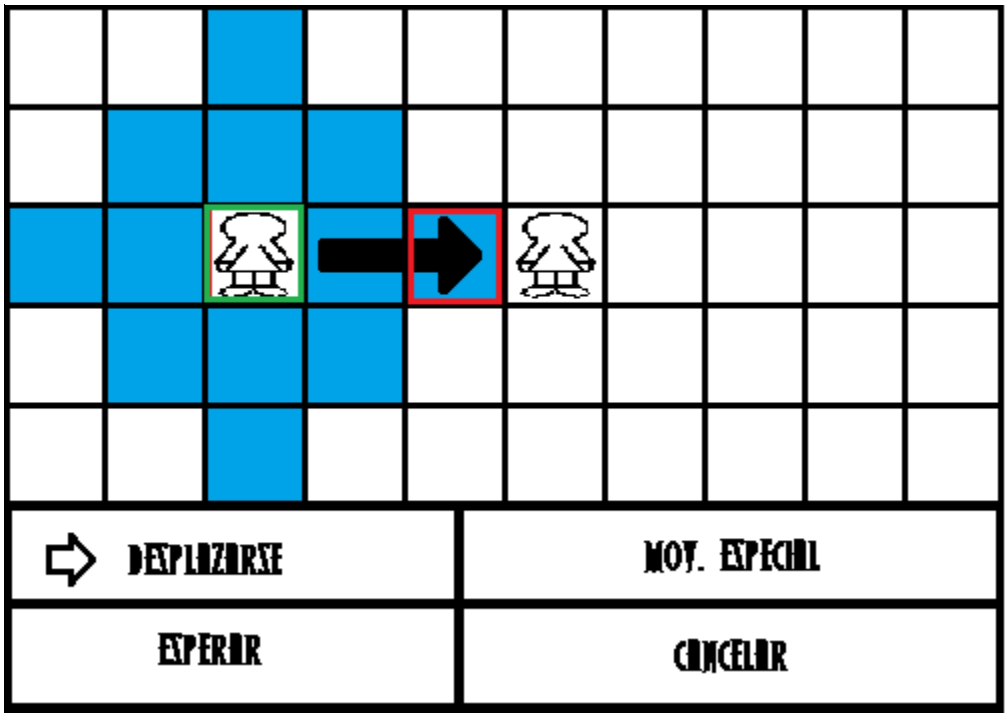
- **Estados alterados de la unidad:**

  Los estados alterados de una unidad puede repercutir en el rango de movimiento, tanto disminuyéndolo como aumentándolo.

- **Terreno del escenario:**

  Hay ciertos tipos de terreno que provocan que la unidad se mueva más lentamente.

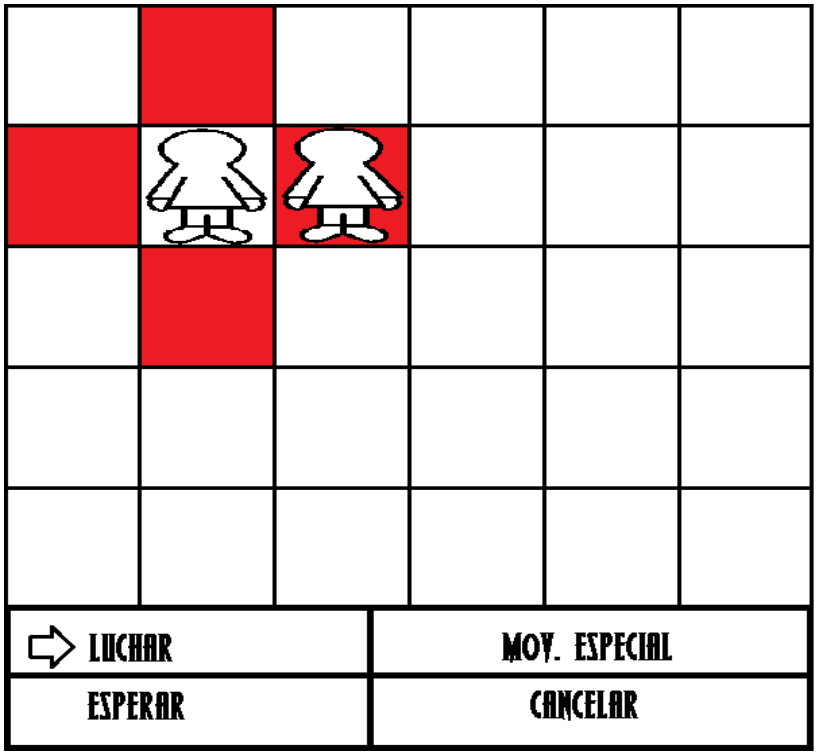
Ilustración 6: Ejemplo se rango de movimiento



Cuando seleccionamos una unidad, también se nos ofrece la posibilidad de comprobar su rango de ataque. Este rango específica a que casillas alcanza su ataque, por lo que si hay un enemigo dentro de las casillas de este rango, podremos atacarlo.

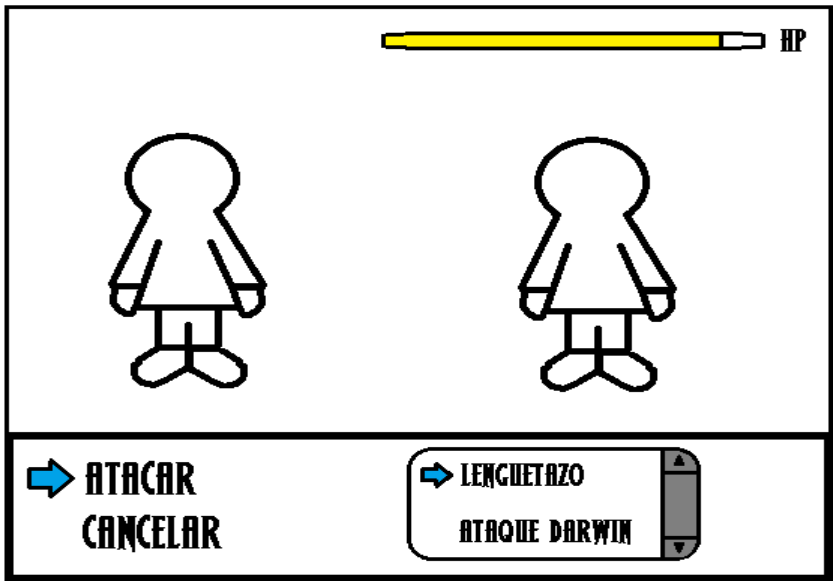
Las distintas tropas, tienen distintos rangos de ataque por lo que se trata de un elemento estratégico de gran importancia.

Ilustración 7: Ejemplo de rango de ataque.



Cuando elegimos luchar, pasamos al modo ataque, donde se muestra los HP de la unidad atacada y un menú táctil donde el usuario puede seleccionar los movimientos que puede realizar.

Ilustración 8: Ejemplo modo batalla.



Cuando el ataque no es a larga distancia, se producirá un intercambio de golpes.

Los marcadores se incrementan a cada turno dependiendo de cuantas centrales energéticas tenga tu bando conquistadas. Para conquistar una central eléctrica, el jugador deberá posicionar una de sus unidades en su casilla y elegir la opción “conquistar”. La conquista puede conllevar varios turnos, dependerá de los HP de los que disponga la unidad (a más HP, menos turnos llevará la conquista).

La radiactividad se incrementa en 40 por cada turno y otros 40 adicionales por cada central nuclear conquistada. En el caso de la energía eólica e hidráulica, se incrementará en 30 por cada central conquistada.

Para vencer, el jugador deberá; o bien acabar con todas las tropas enemigas o conquistar todas las centrales energéticas del tablero.

Con esto, queda explicado de forma básica lo que supone el pilar básico del gameplay.

3.2 Unidades

- **STINKY STING:**

Ataque: 50

Defensa: 50

HP: 30

Desplazamiento: 20

Rango Ataque: 5

Coste: 20 radiactividad

Características especiales:

- Invisible en montaña.

- **DARWIN:**

Ataque: 30

Defensa: 10

HP: 30

Desplazamiento: 30

Rango Ataque: 50

Coste: 20 radiactividad

Características especiales:

- Invisible en agua.

- **MOKELE EMBEMBE:**

Ataque: 80

Defensa: 50

HP: 30

Desplazamiento: 10

Rango Ataque: 5

Coste: 100 radiactividad

Características especiales:

- No puede ser transportando y no es invisible en ningún terreno.

- **PORTALISTA:**

Ataque: 30

Defensa: 30

HP: 20

Desplazamiento: 80

Rango Ataque: 30

Coste: 50 radiactividad

Características especiales:

- Se desplaza a través de portales, lo que le permite llegar a una casilla sin pasar por las colindantes.

- **SCHLACH:**

Ataque: 30

Defensa: 80

HP: 30

Desplazamiento: 30

Rango Ataque: 50

Coste: 100 radiactividad

Características especiales:

- No puede ser transportando y no es invisible en ningún terreno.

- **SUCKY:**

Ataque: 30

Defensa: 30

HP: 20

Desplazamiento: 40

Rango Ataque: 10

Coste: 60 radiactividad, 30 energía eólica

Características especiales:

- Es aéreo, por lo que las unidades terrestres normales no pueden atacarlo.
- Para generarlo, debes tener energía eólica.
- Solo ataca a unidades terrestres.

- **FAN TORNADO:**

Ataque: 40

Defensa: 40

HP: 10

Desplazamiento: 40

Rango Ataque: 10

Coste: 30 radiactividad, 20 energía eólica

Características especiales:

- Es aéreo, por lo que las unidades terrestres normales no pueden atacarlo.
- Para generarlo, debes tener energía eólica.
- Solo ataca a unidades aéreas.

- **TIRAESPINAS:**

Ataque: 40

Defensa: 40

HP: 30

Desplazamiento: 5

Rango Ataque: 20

Coste: 30 radiactividad

Características especiales:

- Solo puede atacar a unidades aéreas.
- Invisible en bosque.

- **BUSS:**

Ataque: 0

Defensa: 40

HP: 40

Desplazamiento: 40

Rango Ataque: 0

Coste: 30 radiactividad

Características especiales:

- Puede transportar 2 unidades.
- Hace visibles a las unidades ocultas enemigas.

- **TOXIC FLAKE:**

Ataque: 30

Defensa: 30

HP: 30

Desplazamiento: 20

Rango Ataque: 30

Coste: 60 radiactividad, 30 energía hidráulica.

Características especiales:

- Invisible en agua.
- Solo ataca a unidades terrestres.
- Es acuático, no puede ser atacado por unidades terrestres.
- Se necesita energía hidráulica para crearlos.

- **SUBSNAKE:**

Ataque: 40

Defensa: 40

HP: 30

Desplazamiento: 20

Rango Ataque: 10

Coste: 30 radiactividad, 20 energía hidráulica.

Características especiales:

- Ataca solo unidades acuáticas.
- Invisible en agua.

- **ELECTRO WOLF:**

Ataque: 40

Defensa: 40

HP: 30

Desplazamiento: 5

Rango Ataque: 20

Coste: 30 radiactividad

Características especiales:

- Solo puede atacar a unidades acuáticas.
- Invisible en bosque.

3.3 Marcadores

Existen tres marcadores determinantes en la fabricación de unidades; radiactividad, energía eólica y energía hidráulica.

La radiactividad será necesaria para crear cualquier unidad y para ejecutar algunos movimientos especiales. Se incrementa automáticamente después de cada turno proporcionalmente al número de centrales nucleares que el jugador tenga conquistadas.

La energía eólica será necesaria para crear unidades aéreas. Se incrementa automáticamente después de cada turno proporcionalmente al número de centrales eólicas que el jugador tenga conquistadas.

La energía hidráulica será necesaria para crear unidades marinas. Se incrementa automáticamente después de cada turno proporcionalmente al número de centrales

hidráulicas que el jugador tenga conquistadas, estas centrales siempre están establecidas en áreas acuáticas.

3.4 Tipos de Terreno

Los tipos de terreno son un elemento que el jugador debe tener en cuenta a la hora de elaborar sus estrategias. Las unidades son favorecidas o perjudicabas por los distintos tipos, porque habrá que tener cuidado con determinadas zonas.

- **Tierra**: Terreno más común. Terreno completamente neutro. No proporciona ventajas o desventajas a ninguna unidad.
- **Montaña**: Proporciona la posibilidad de ocultarse a Stinky Sting.
- **Agua**: Proporciona la posibilidad de ocultarse a Darwin, Subsnake, y aToxic Flake.
- **Bosque**: Proporciona la posibilidad de ocultarse a Tiraespinas y a Electro Wolf.

3.5 Movimientos especiales

Son movimientos que un "P.U Pet" puede realizar y que no suponen un ataque directo al enemigo, es decir, que no suponen una reducción inmediata de los HP del enemigo. Tienen gran valor estratégico, pero al realizarlos la unidad perderá turno, lo que implica que no se puede mover ni atacar:

- **Mina Radioactiva**: Contamina una casilla de manera invisible. Si el enemigo se coloca sobre una casilla contaminada, su estado pasará a ser "Contaminado". Consume radioactividad.
- **Intimidar**: Intimida a un enemigo. Su estado pasará a ser "Asustado".
- **Gas Tóxico**: Contamina un área de varias casillas de forma visible. Si una unidad entra en contacto con esta área, su estado pasará a ser "Envenenado".
- **Reunir Valor**: La unidad que lo realice cambiará su estado a "Envalentonado".
- **Sentir**: Descubre la posición de las unidades ocultas en el área establecida (+4 casillas en todas direcciones de la posición actual).
- **Ocultarse**: Solo disponible cuando el terreno es propicio para la unidad. Hace invisible al enemigo a la tropa. Consume radiactividad. La invisibilidad de las unidades se anula cuando otra unidad entre en su área o cuando ataca.

- **Deprisa**: Cambia el estado de la unidad a “Prisa”.
- **Aliviar**: Aumenta en 2 puntos los HP de la unidad cada turno.
- **Curar**: Aumenta 15 HP a la unidad seleccionada.
- **Sosegar**: Devuelve el estado normal a la unidad seleccionada. Puede ser también a unidades enemigas.

3.6 Estados Alterados

Provocan una alteración en las estadísticas de un “P.U Pet”. Pueden alterar su rango de ataque, su rango de movimiento o puede suponer una pequeña perdida de HP por turno.

Ejemplo de estados alterados:

- **Envenenado**: La unidad pierde 2 HP por turno hasta que su estado vuelva a ser normal.
- **Asustado**: Reduce el rango de movimiento.
- **Cegado**: Reduce el rango de ataque.
- **Envalentonado**: Aumento el rango de ataque.
- **Prisa**: Aumenta el rango de movimiento.
- **Contaminado**: La unidad pierde 2 HP y reduce su rango de movimiento por turno hasta que su estado vuelva a ser normal.

3.7 Historia y Ambientación

P.U Pets, será un proyecto centrado en la jugabilidad y donde premia la simpleza por lo que no tendrá una historia narrada. Solo se mostrará el contexto y se darán pistas al jugador de lo sucedido.

Historia:

Una mega-corporación llamada “WastedPeople Inc.”. Tenía como actividad principal el desarrollo de productos tecnológicos, sin embargo, fue pionera en la creación de mascotas diseñadas a través de la ingeniería genética.

Comenzó la venta de los llamados “P.U Pets”, mascotas cuya finalidad era el entretenimiento de sus dueños, completamente autos suficientes, lo que los hacía mucho más atractivos al consumidor que los animales convencionales.

Para su elaboración, se crearon las llamadas “madres”, dos Pets, capaz de generar por si mismos las unidades que se ponían a la venta, ahorrándose de esta manera mano de obra. También era necesaria cierta cantidad de radiactividad, por lo que se le dieron uso a los residuos radiactivos generados por las centrales durante su actividad, subsanando así uno de los mayores problemas de este tipo de energía.

Esto provocó que la energía nuclear proliferara y se posicionará como la principal en todo el mundo. Sin embargo, su naturaleza inestable hizo acto de presencia y provocó el mayor accidente en cadena de la historia acabando con la vida de miles de millones de personas.

La radiactividad, acabó por contaminarlo todo, lo que provocó la completa extinción de la raza humana y la proliferación de los P.U Pets. Las dos “madres” formaron dos facciones enfrentadas por tomar el control de cuantas centrales nucleares les fuera posible, con el propósito de expandir sus creaciones.

Narración:

A pesar de tener una historia consistente para el proyecto, no habrá una narración como tal en el juego. La historia se dará a conocer al jugador a través de carteles publicitarios que se pueden ver por el escenario. Esto servirá para que el jugador forme sus propias teorías y ponga atención en los elementos del escenario en busca de nuevas pistas. Nos serviremos del sentido del humor y de la ironía para poner de manifiesto la inhumanidad de las mega-corporaciones.

Ambientación:

El juego transcurre en un mundo pos-apocalíptico llamado “Mundo Baldío”. Todo está arrasado y solo quedan en pie las centrales energéticas que se disputan las dos facciones de P.U Pets enfrentadas. En los escenarios primaran los tonos pastel, dando a los escenarios un aspecto desértico que contrastará con el colorido de los Pets, que tienen aspecto de juguetes o mascotas. Aunque también habrá zonas boscosas, montañosas y acuáticas, las zonas más comunes serán las zonas de tierra.

3.8 Diseño GUI

La GUI o interfaz gráfica de usuario de P.U Pets, se aprovechará de la pantalla táctil de los dispositivos móviles. Utilizaremos solo interacción mono táctil por dos motivos; sencillez y hacer el juego compatible con mayor número de móviles.

Este segundo propósito es el motivo por el que haremos el videojuego compatible con una gran amplitud de resoluciones con el objetivo de hacerlo compatible con las pantallas de diversos dispositivos. Habrá que detectar estas posibles resoluciones y adaptar los sprites animados y escenarios.

Como ha quedado explicado en el apartado de "Mecánica jugable", el jugador se moverá por el tablero/escenario a través de una interfaz gráfica que simulará un joypad con 4 direcciones y un botón táctil. El motivo de no hacerlo directamente a través de simples toques en el cuadrante que se quieres seleccionar, es que serían necesarios cuadrantes demasiado grandes como para poder mostrar suficiente mapeado, lo cual podría resultar incomodo para el jugador. Sin embargo, los menús contextuales, si serán puramente táctiles, por los que habrá que hacerlos suficientemente grandes como para poder seleccionarlos con un toque táctil de manera sencilla.

La HUD, en el modo tablero estará compuesta por los distintos marcadores energéticos (radiactividad, energía eólica y energía hidráulica) además de un indicador de unidades disponibles. En el modo batalla, se indicará los HP de las unidades enfrentadas, y con cada ataque se marcará cuantos se les resta a la unidad dañada.

4 ANALISIS TÉCNICO

4.1 Interacción

Como ha quedado definido en el apartado "Diseño GUI", la interacción principal entre el usuario y el videojuego será a través de la pantalla táctil. Por la sencillez de la interfaz de usuario, solo se interaccionará a través de gestos mono-táctiles lo que a su vez posibilitará mayor rango de compatibilidad con dispositivos móviles. No se hará ningún uso del teclado.

4.2 Versiones Android compatibles

Como hemos comentado anteriormente, uno de los objetivos de este proyecto es hacer un videojuego asequible, que llegue al mayor número de gente posible. Para ello es primordial hacer que P.U Pets sea muy escalable y sea compatible con el mayor número de dispositivos móviles posible.

Esta decisión no tendría sentido si no se tuviese en cuenta las distintas características de las sucesivas versiones de Android. Por lo tanto, haremos compatible el proyecto con un amplio rango de versiones.

Rango de Versiones compatibles: Android 2.1 (Éclair) – Android 4.2 (Jelly Bean).

4.3 Permisos y características

Nuestro Framework necesitará un archivo declarativo donde queden definidas las características principales del proyecto, tales como los permisos necesarios o la orientación de la aplicación.

En nuestro XML, quedarán definidas las siguientes características:

- screenOrientation → landscape: Con el propósito de mejorar la visibilidad del campo de batalla, usaremos el modo apaisado fijo en la pantalla.

- android.permission.WAKE_LOCK: Con este permiso, nos aseguramos de que el dispositivo móvil no entre en modo reposo durante la ejecución de nuestro videojuego.

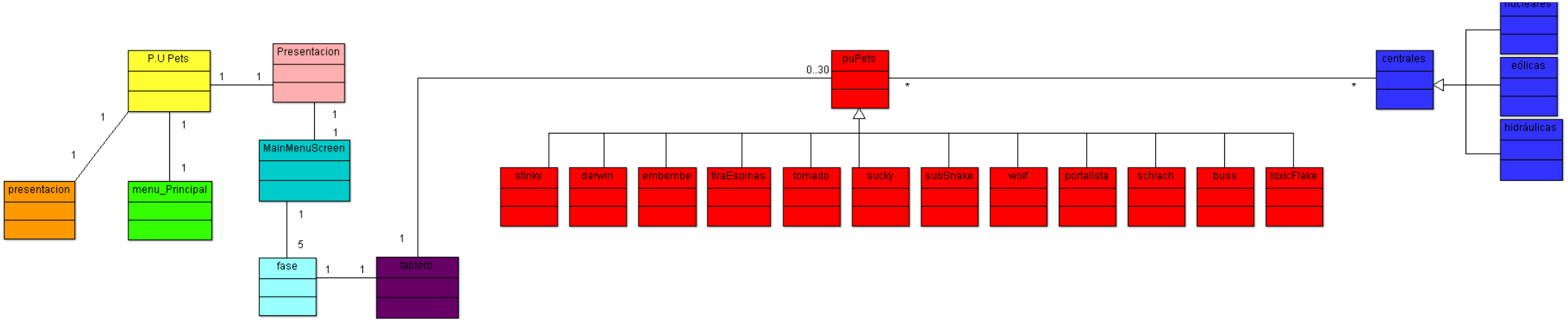
- Android.permission.WRITE:EXTERNAL_STORAGE: Usaremos este permiso para escribir o leer archivos ubicados en una unidad externa de almacenamiento, en este caso la micro-SD con la que suelen contar los móviles. Esto será crucial para tereas de tanta importancia como por ejemplo guardar la partida.

5 DISEÑO TÉCNICO

Lo que se expone a continuación son los diagramas técnicos que corresponderías al código del proyecto. Se corresponde con el diseño previo y podrían producirse cambios durante la codificación.

5.1 Diagrama de clases de lógica de negocio

Ilustración 9: Diagrama de clases planificado.

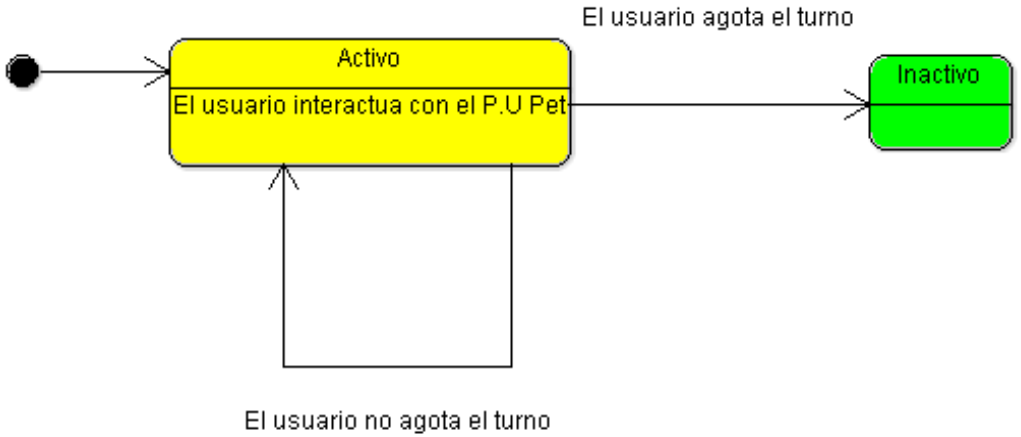


5.2 Diagrama de estados

Estados de los objetos "puPets":

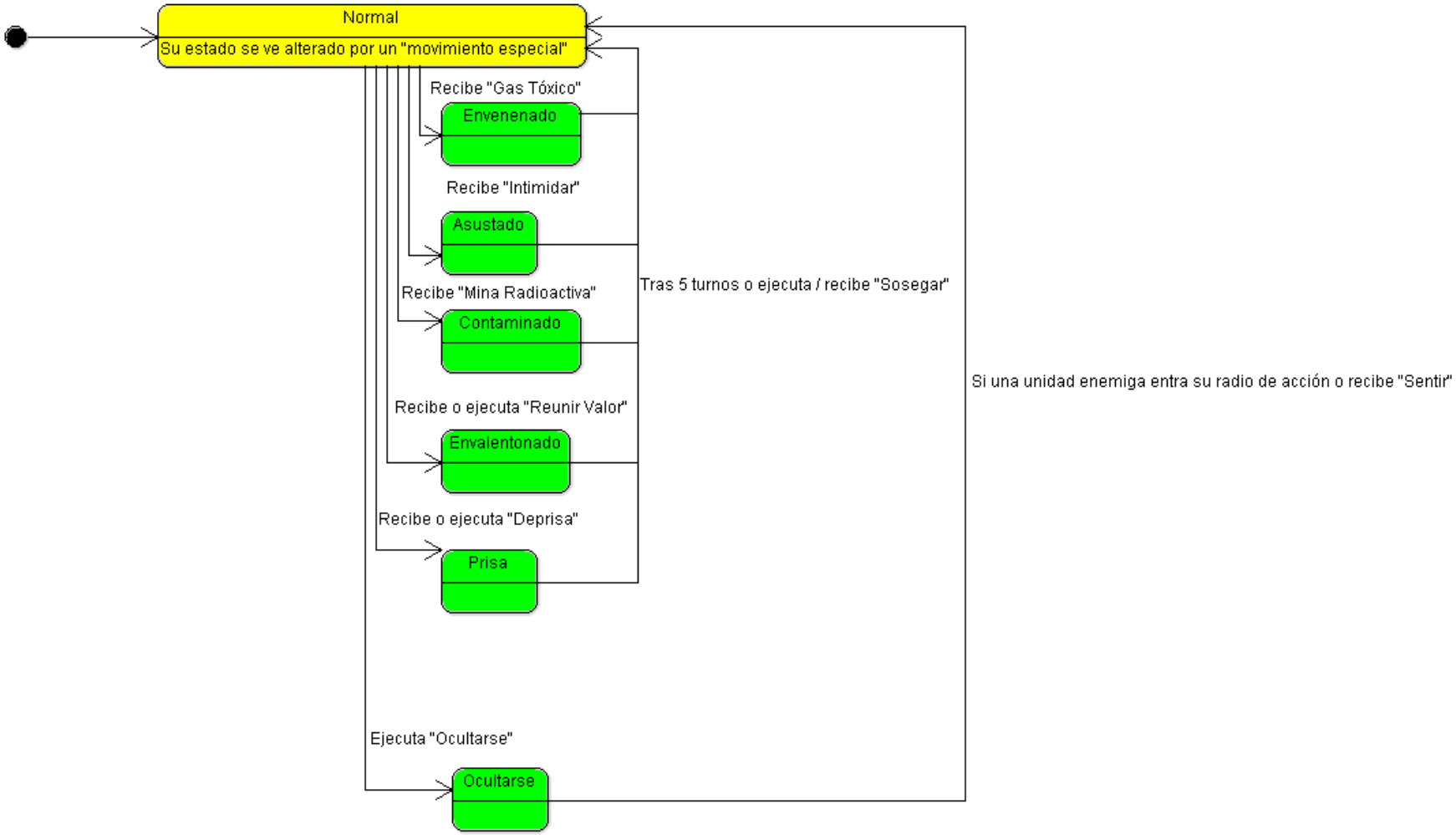
- Relativo a la "actividad":

Ilustración 10: Diagrama de estados relativo a los turnos.



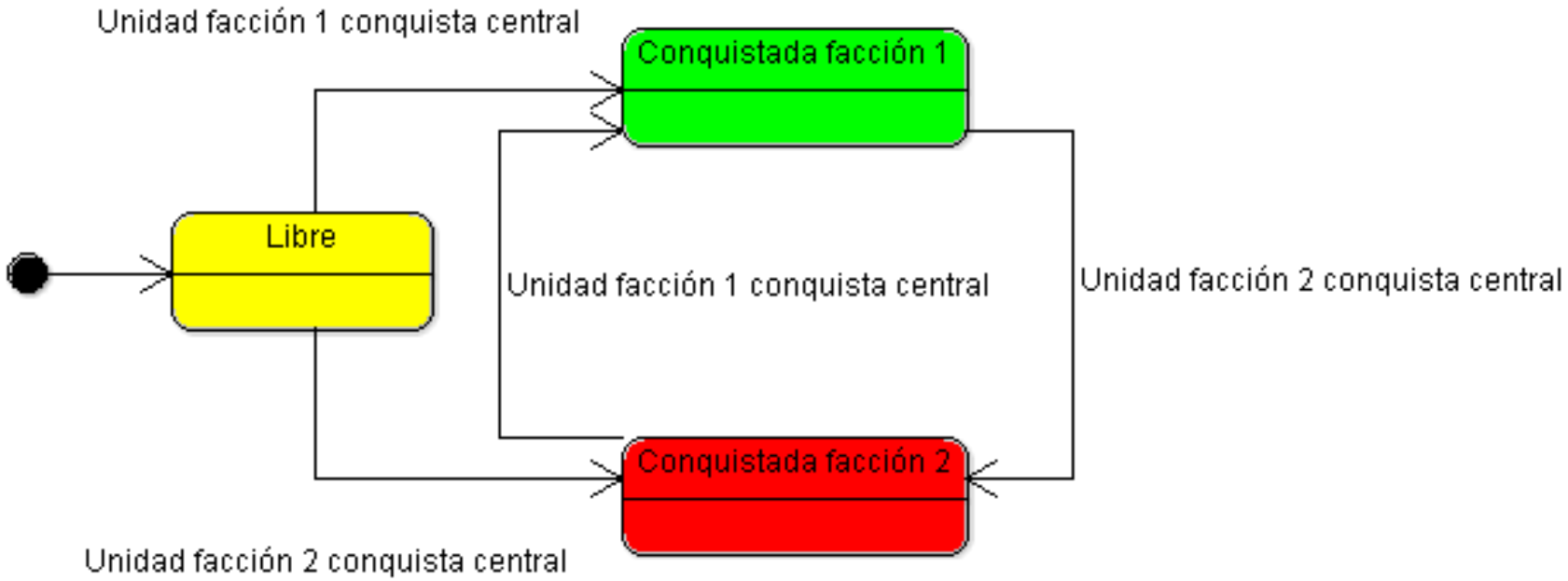
- Relativo a los “estados alterados” de los objetos puPets:

Ilustración 11: Diagrama de estados relativo a los estados alterados.



Estados de los objetos “centrales”:

Ilustración 12: Estados de las centrales energéticas.



6 IMPLEMENTACIÓN DEL FRAMEWORK

Con motivo de poder reutilizar el framework para futuros proyectos sin tener que hacerle demasiados añadidos, hemos implementado clases controladoras de todos los dispositivos de entrada disponibles en los móviles. De esta manera, dispondremos de un framework operativo, lo que nos permitirá ahorrar tiempo de desarrollo.

El denominado "Raconned Framework", se compone de dos paquetes; com.raconned.framework donde se encuentran las interfaces a implementar y com.raconned.framework.impl donde se encontraran estas implementaciones. Esta sería su estructura:

com.raconned.framework

- Audio.java
- FileIO.java
- Game.java
- Graphics.java
- Input.java
- Music.java
- Pixmap.java
- Pool.java
- Screen.java
- Sound.java

com.raconned.framework.impl

- AccelerometerHandler.java
- AndroidAudio.java
- AndroidFastRenderView.java
- AndroidFileIO.java
- AndroidGame.java
- AndroidGraphics.java
- AndroidInput.java
- AndroidMusic.java
- AndroidPixmap.java
- AndroidSound.java
- KeyboardHandler.java
- MultiTouchHandler.java
- SingleTouchHandler.java
- TouchHandler.java

Pasemos a comentar la implementación lógica de las distintas clases:

6.1 AccelerometerHandler

Esta clase nos sirve para controlar los cambios producidos en el acelerómetro del dispositivo móvil. Se vale de las librerías Sensor, e implementa la interfaz SensorEventListener, para detectar cambios en el estado de cualquier eje del acelerómetro.

Métodos implementado:

- void onSensorChanged(SensorEvent event): Sirve para almacenar los valores de los ejes que tiene el acelerómetro.

- float getAccelX(): Revuelve la aceleración del eje X.
- float getAccelY(): Revuelve la aceleración del eje Y.
- float getAccelZ(): Revuelve la aceleración del eje Z.

6.2 AndroidAudio

Implementa la interfaz Audio.java con el propósito de servirnos unos métodos de carga de audio (tanto sonidos como canciones) desde archivos localizados en los Assets . Métodos implementados:

- Music newMusic(String filename): Carga un archivo de música desde los assets. En el caso de no encontrar el archivo, lanzará un RuntimeException, donde se mostrará el nombre del archivo que ha provocado el error.
- Sound newSound(String filename): Carga un archivo de sonido desde los assets. En el caso de no encontrar el archivo, lanzará un RuntimeException, donde se mostrará el nombre del archivo que ha provocado el error.

6.3 AndroidFastRenderView

Se vale de una instancia de Game para obtener el Screen que se está mostrando para dibujar de forma constante. Para ello dibujará lo contenido en el framebuffer (de la instancia AndroidGraphics) recalándolo en el caso de que haya diferencia entre las dimensiones del rectángulo del Canvas y el propio framebuffer de la instancia de Game.

Métodos implementados:

- void resume(): Se encarga de iniciar el hilo de ejecución.
- void run(): Crea el bucle principal del juego. También es el encargado de las labores de recalado de los gráficos.
- void pause(): Se encarga de para el hilo de ejecución. Una vez detenido, puede devolver el control.

6.4 AndroidFileIO

Implementa la interfaz FileIO.java para proporcionar métodos de E/S de datos a través de ficheros. Lanzará una excepción del tipo IOException en el caso de no poder acceder a alguno de los archivos tratados.

Métodos implementados:

- InputStream readAsset(String fileName): Se encarga de la lectura de un fichero almacenado en Assets. Lanzará una excepción del tipo IOException en el caso de no poder acceder al archivo.

- InputStream readFile(String fileName): Se encarga de la lectura de un fichero almacenado en una unidad externa. Lanzará una excepción del tipo IOException en el caso de no poder acceder al archivo.
- OutputStream writeFile(String fileName): Se encarga de la escritura en un fichero almacenado en una unidad externa. Lanzará una excepción del tipo IOException en el caso de no poder acceder al archivo.

6.5 AndroidGame

Implementa la interfaz Game.java con el propósito de gestionar la actividad y el estado de la misma, así como la vista e iniciar las instancias necesarias para ejecutar el juego. Además se encarga de evitar que la pantalla pase a modo ahorro de energía.

En esta clase, es donde asignamos unos valores a las dimensiones del framebuffer, en este caso se han tomado como dimensiones de pantalla por defecto 800x480.

Métodos implementados:

- void onCreate(Bundle savedInstanceState): Sirve para iniciar la Activity. En este método, se crean instancias y se inicializan con valores por defecto.
- void onResume(): Se utiliza para iniciar el Wakelock, para que el dispositivo móvil no pueda pasar a modo de ahorro de energía y para que AndroidFastRenderView inicie el hilo de ejecución principal.
- void onPause():Se utiliza para parar el Wakelock, para que el dispositivo móvil no pueda pasar a modo de ahorro de energía y para que AndroidFastRenderView pare el hilo de ejecución principal.
- Input getInput(): Devuelve la instancia de Input.
- FileIO getFileIO(): Devuelve la instancia de FileIO.
- Graphics getGraphics(): Devuelve la instancia de Graphics.
- Audio getAudio():Devuelve la instancia de Audio.
- void setScreen(Screen screen): Asigna a Screen la instancia parada como parámetro, tras haber borrado la actual.
- Screen getCurrentScreen(): Devolverá la pantalla Screen actualmente activa.
- Context getAppContext(): Devuelve el contexto actual.

6.7 AndroidGraphics

Implementa la interfaz Graphics.java para proporcionar métodos para la carga de Bitmaps en distintos formatos (RGB565, ARGB4444, ARGB8888) y su dibujado.

Métodos implementados:

- Pixmap newPixmap(String fileName, PixmapFormat format): Carga un Bitmap en el formato especificado como parámetro. En el caso de fallar la carga, se lanzará una excepción RuntimeException.
- void clear(int color): Colorea el framebuffer del color especificado como parámetro eliminando cualquier contenido.
- void drawPixel(int x, int y, int color): Dibuja un pixel. Su color y sus coordenadas serán las especificadas como parámetros.
- void drawText(String text, int x, int y, Paint paint): Dibuja un texto. Su contenido, coordenadas y fuente (se declarara como paint.setTypeface) serán las especificadas como parámetros.
- void drawLine(int x, int y, int x2, int y2, int color): Dibuja una linea que ira desde las coordenadas x e y hasta las definidas por las coordenadas x2 e y2. El color será el especificado por parámetro.
- drawPixmap(Pixmap pixmap, int x, int y, int srcX, int srcY, int srcWidth, int srcHeight): Dibuja solo una parte de un Pixmap. Las dimensiones de dicha parte hay que especificarla a través de los parámetros (desde el punto (x,y) hasta el punto (srcX,srcY) con la altura srcHeight y la anchura srcWidth).
- void drawPixmap(Pixmap pixmap, int x, int y): Dibuja un pixmap a partir de las coordenadas especificadas por parametro.
- int getWidth(): Devuelve la anchura del framebuffer.
- int getHeight(): Devuelve la altura del framebuffer.

6.8 AndroidInput

Permite controlar eventos relacionado con los dispositivos de entrada, es decir, acelerómetro, pantalla táctil y teclado, valiéndose de instancias de sus respectivas clases controladoras (accelHandler, keyHandler y touchHandler) para hacerlo de manera más sencilla y manejable. En el caso de la interacción con la pantalla táctil, también se encarga de comprobar si la versión de Android en la que estamos ejecutando el programa, es compatible con los toques multi-tactiles, y los imposibilita de no ser así.

Métodos implementados:

- boolean isKeyPressed(int keyCode): Devuelve true si la tecla facilitada por parámetro ha sido pulsada.
- boolean isTouchDown(int pointer): Devuelve true si la pantalla táctil ha sido tocada. El parámetro sirve para detectar múltiples toques simultáneos, a los que se les asigna una ID.

- int getTouchX(int pointer): Devuelve la coordenada x del toque táctil especificado a través de su ID como parámetro.
- int getTouchY(int pointer): Devuelve la coordenada y del toque táctil especificado a través de su ID como parámetro.
- float getAccelX(): Devuelve la aceleración en el eje x del acelerómetro.
- float getAccelY(): Devuelve la aceleración en el eje y del acelerómetro.
- float getAccelZ(): Devuelve la aceleración en el eje z del acelerómetro.
- List<KeyEvent> getKeyEvents(): Devuelve el listado de eventos acontecidos en el teclado táctil.
- List<TouchEvent> getTouchEvents(): Devuelve el listado de eventos acontecidos en la pantalla táctil.

6.9 AndroidMusic

Proporciona métodos que permiten interactuar con archivos de música previamente cargados. Estos métodos nos permiten desde reproducir/parar la canción, ponerla en pausa, establecer la reproducción de la música en modo "looping, establecer un volumen de reproducción, hasta comprobar el estado de la reproducción.

Métodos implementados:

- void onCompletion(MediaPlayer arg0): Se vale de la sincronización de un bloque para informar correspondientemente el booleano isPrepared.
- void play(): Sirve para reproducir la pista cargada previamente.
- void stop(): Sirve para parar la reproducción de la pista cargada previamente.
- void pause(): Si la pista se está reproduciendo, la detiene. Si esta parada, la reproduce.
- void setLooping(boolean looping): Establece el modo looping en la reproducción, según se indique reciba true o false del parámetro.
- void setVolume(float volume): Establece como volumen de la reproducción el valor introducido como parámetro.
- boolean isPlaying(): Devuelve true o false dependiendo de si la reproducción de la pista está en curso.
- boolean isStopped(): Devuelve true o false dependiendo de si la reproducción de la pista está detenida.
- boolean isLooping(): Devuelve true o false dependiendo de si la reproducción de la pista esta en modo looping o no.
- void dispose(): Detiene la reproducción en curso en caso de haberla, y libera los recursos requeridos por la carga de la pista.

6.10 AndroidPixmap

Implementación de la interfaz Pixmap.java. Contiene métodos que proporcionan información sobre un gráfico determinado.

Métodos implementados:

- int getWidth(): Devuelve la anchura del Pixmap .
- int getHeight(): Devuelve la altura del Pixmap.
- PixmapFormat getFormat(): Devuelve el formato del Pixmap.
- void dispose(): Libera los recursos requeridos por la carga del Pixmap.

6.11 AndroidSound

Implementa la interfaz Sound.java con el propósito de reproducir archivos de sonido previamente cargados.

Métodos implementados:

- void play(float volume): Reproduce la pista de sonido cargada con el volumen especificado como parámetro.
- void dispose(): Libera los recursos requeridos por la carga del sonido.

6.12 KeyboardHandler

Se encarga de la recepción y procesado de eventos provenientes del teclado (a través de la vista View). Permite comprobar si una tecla del teclado virtual esta pulsada o cual es esta tecla.

Métodos implementados:

- boolean onKey(View v, int keyCode, android.view.KeyEvent event): Procesa los eventos sucedidos en el teclado recibidos a través de la vista especificada como parámetro.
- boolean isKeyPressed(int keyCode): Devuelve true si la tecla cuyo código se especifica como parámetro, esta pulsada.
- List<KeyEvent> getKeyEvents(): Devuelve el listado de eventos almacenado recibidos a través del teclado táctil.

6.13 MultiTouchHandler

Se utilizará para la recepción y procesado de eventos provenientes de la pantalla táctil, en el caso de trabajar con pantallas multitáctiles.

Métodos implementados:

- boolean onTouch(View v, MotionEvent event): Procesa los eventos sucedidos en la pantalla táctil recibidos a través de la vista especificada como parámetro. Detecta los siguientes eventos:
    - TOUCH_DOWN: Pantalla pulsada.
    - TOUCH_UP: Pulsación terminada.
    - TOUCH_DRAGGED: Arrastre por la pantalla táctil.

- boolean isTouchDown(int pointer): Devuelve true si el evento táctil al que corresponde índice pasado como parámetro es un toque táctil.
- int getTouchX(int pointer): Devuelve la coordenada x del evento táctil al que corresponde índice pasado como parámetro.
- int getTouchY(int pointer): Devuelve la coordenada y del evento táctil al que corresponde índice pasado como parámetro.
- List<TouchEvent> getTouchEvents(): Devuelve el listado de eventos almacenado recibidos a través de la pantalla táctil.

6.14 SingleTouchHandler

Se utilizará para la recepción y procesado de eventos provenientes de la pantalla táctil, en el caso de trabajar con pantallas monotáctiles, o en el caso de no ser necesarios gestos multitáctiles.

Comparte métodos con MultiTouchHandler, al implementar ambos la interfaz TouchHandler.

6.15 TouchHandler

Se trata de una interfaz que hereda de OnTouchListener y será utilizada tanto de SingleTouchHandler como MultiTouchHandler.

6.16 Pool

Sirve para reutilizar instancias ya creadas cada vez que haya que almacenar un evento de teclado o de pantalla táctil, para evitar que el recolector de basura la elimine.

7 DESARROLLO

Tras planificar el proyecto, se comienzan las labores de desarrollo, teniendo en cuenta, que es posible que algunos de los aspectos definidos durante esta etapa, puedan quedar fuera del proyecto final por falta de tiempo o por no pasar las pruebas de diseño. Estas decisiones serían tomadas buscando la máxima eficiencia en el aprovechamiento del poco tiempo disponible y procurando una jugabilidad bien equilibrada.

Se ha procurado un código robusto, sin errores críticos, pero aún así habría que procurarle una fase de testeo más amplia, para evitar la aparición de los indeseados bugs y glitches.

Para el correcto funcionamiento del videojuego, ha sido necesaria la implementación de varios algoritmos muy poco conocidos dentro del ámbito de la industria de los videojuegos, como por ejemplo la matriz de tiles. En este documento se pondrá especial atención a estos algoritmos, para que su funcionamiento quede perfectamente explicado.

A continuación, son desglosados los métodos más críticos del desarrollo en su versión final.

7.1 ALGORITMO DE GENERACIÓN DE ESCENARIOS A PARTIR DE UNA MATRIZ DE TILES

Primero habría que explicar el concepto de tile. Un tile es algo así como una baldosa. Consiste en un valor numérico dentro de una matriz, que indicaría al código que tipo de baldosa hay que poner y donde. Por lo tanto, los escenarios se forman a partir de un algoritmo, que tomando como referencia la matriz de tiles, los va construyendo baldosa a baldosa.

Las ventajas de la utilización de este algoritmo son muchas. Por ejemplo, se ahorra mucha memoria, puesto que los escenarios se construyen a partir de un pequeño número de gráficos de pequeño tamaño ya cargados en el framebuffer (en el caso de este proyecto, son solo 4 tipos de gráficos para el escenario más otros 3 de las centrales energéticas). También agiliza la creación de distintos escenarios para distintas fases, ya que basta con cambiar la matriz de tiles para generar un escenario completamente nuevo.

En el código del presente proyecto, en este cometido toman partido las clases “World.java”, “Tablero.java” y “Tablero_FaseX.java”.

World, es la clase que contiene la lógica del videojuego, así que será el encargado principal de la generación del escenario. Para ello, se vale de instancias de Tablero, que contiene la lógica del escenario y de Tablero_FaseX, siendo la X el número de fase que se está ejecutando, y que es la encargada de informar con los valores precisos la matriz de tiles para crear el escenario correspondiente a la fase.

Para llevarlo a cabo, la clase World, contiene el método drawFields con la siguiente firma:

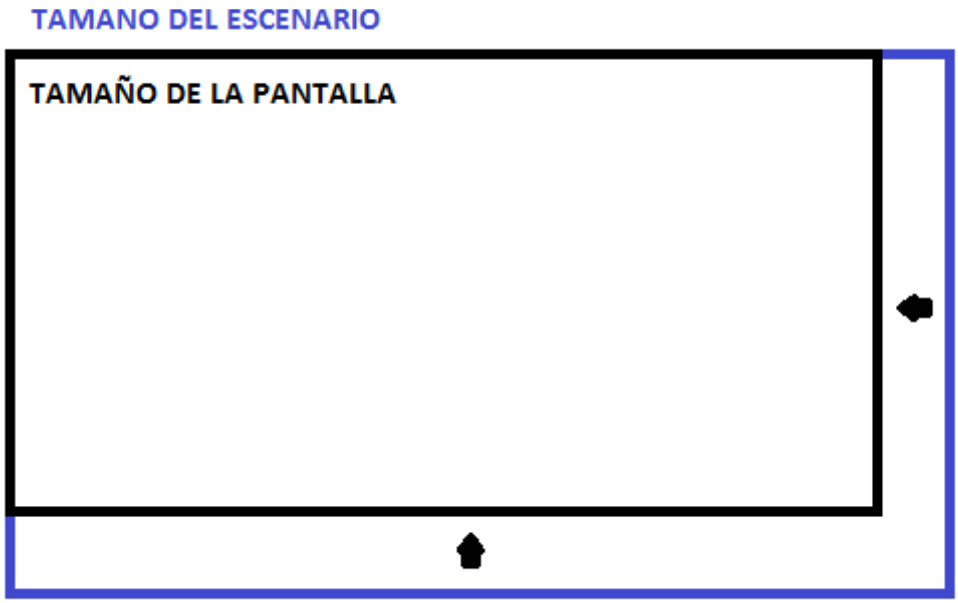
- public void drawFields(int j_ini, int i_ini).

Este método, construirá el escenario a partir de la matriz de tiles definida en Tablero_FaseX que para dibujarlo en pantalla en el método present.

Los dos enteros que recibe como parámetro, sirve para conocer la posición relativa del cursor de juego* (la instancia de Point.java) en relación a la pantalla. Solo serán útiles en caso de que la matriz de tiles, sea más grande de lo que se puede mostrar en pantalla. En este caso, la pantalla solo mostrará una parte del escenario y en caso de que el jugador quiera llegar a las partes no mostradas del escenario habrá que hacer lo que se conoce como scroll.

El scroll es el desplazamiento del escenario para dar sensación de movimiento y en el caso de los juegos en 2D, puede ser vertical u horizontal

Ilustración 13: Ejemplo de encuadre del scroll



En el proyecto que nos ocupa, en la pantalla se van mostrando las partes del escenario según las partes de la matriz de tiles que se pueden mostrar. Para saber, de donde a donde hay que dibujar, el método drawFields se vale de sus parámetros ini_j e ini_i y de las constantes rowMaxScreen y colMaxScreen que marcan el número de filas y columnas máximas que caben en la pantalla. El método simplemente se encarga de dibujar la parte de la matriz de tiles requerida, posibilitando así el desplazamiento por el escenario.

Ilustración 14: Pantalla con ini_j a 0

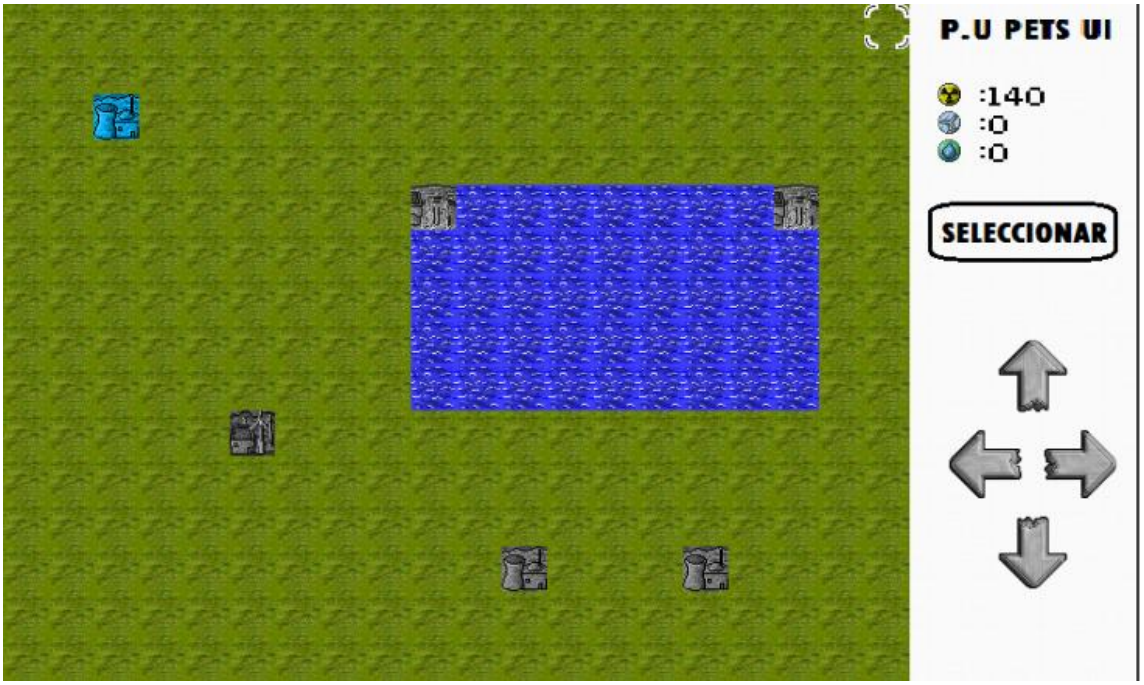
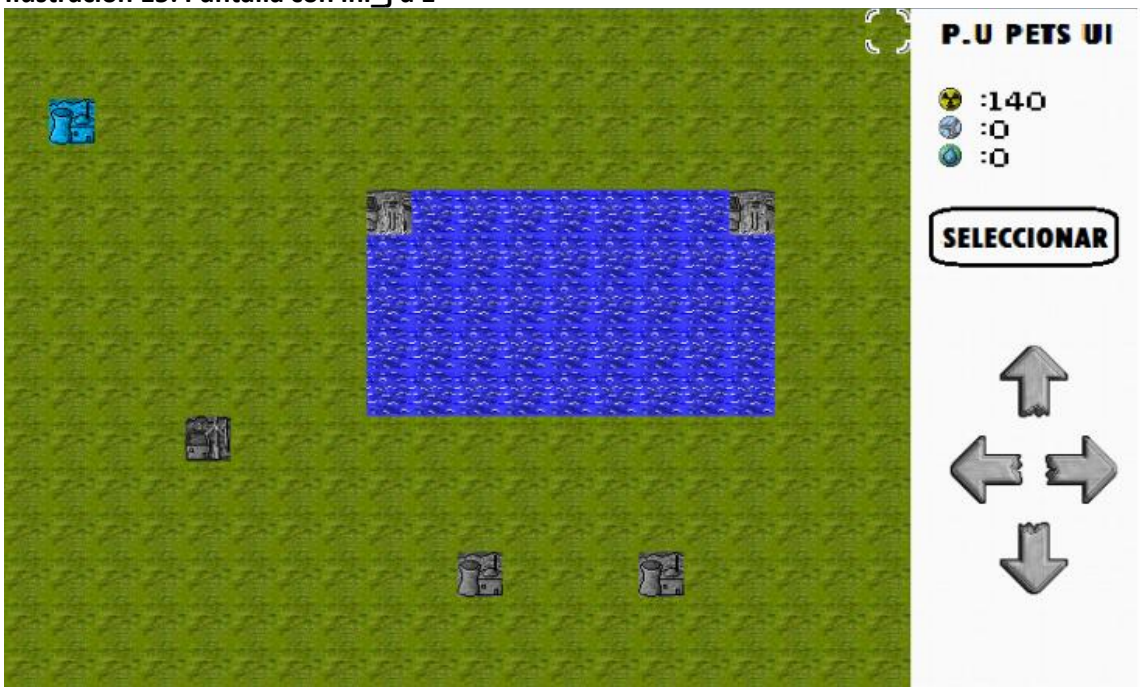


Ilustración 15: Pantalla con ini_j a 1



El encargado de informar con sus valores correspondientes a ini_j e ini_i es el método present, donde se detecta la posición del cursor de juego, y si esta fuera de los límites de lo que la pantalla puede mostrar, se le suma uno. De esta manera, si estamos mostrando de la fila 0 a al rowMaxScreen y el jugador mueve el cursor a una fila mayor que la constante, ini_j pasará a valer 1 y el método drawFields construirá el escenario a partir de la fila 1 hasta la fila rowMaxScreen+1, dando sensación de desplazamiento.

La traducción de los valores de la matriz de tiles a gráficos, tendría la siguiente correspondencia:

Casilla a 0: Tierra. Se carga el siguiente tile: 

Casilla a 1: Bosque. Se carga el siguiente tile: 

Casilla a 2: Agua. Se carga el siguiente tile: 

Casilla a 3: Montaña. Se carga el siguiente tile: 

Casilla a 4: Central. Detecta a través de un array que almacena objetos de tipo Energy_Plant, de que bando es (usuario, enemigo) y de qué tipo (nuclear, hidráulica o eólica) para mostrar su gráfico correspondiente.

7.2 ALGORITMO DE FORMACIÓN DEL RANGO DE MOVIMIENTOS Y ATAQUE

En el código podemos comprobar que la matriz de tiles, es de tres dimensiones. Esto se debe a que se usa cada dimensión con un cometido.

La primera dimensión es la encargada de almacenar los valores de construcción del escenario como hemos visto en el apartado anterior.

La segunda dimensión es la encargada de marcar el rango de movimiento de los pets. Cuando el usuario solicita el desplazamiento de una de sus unidades, las casillas por las que se puede mover se marcan en azul. Esto se consigue gracias al método drawRangoMove de la clase Tablero, que construye el rango de movimiento informando las casillas de la segunda dimensión de la matriz que se englobarían dentro de este rango a 1, dependiendo del tipo de pet del que se ha solicitado el movimiento, su estadística "desplazamiento" y del tipo de terreno que contiene la matriz en la casilla de la primera dimensión.

Ilustración 16: Rango de ataque

La misma lógica es aplicada al rango de ataque, con la salvedad de que en este caso, para evitar solapamientos, se usa la tercera dimensión de la matriz. Las casillas englobadas dentro del rango de ataque de un pet, quedarán marcadas en rojo una vez solicitado por el usuario.

Ilustración 17: Rango de ataque

Puede sonar complejo, pero en realidad no lo es. Vayamos por partes.

La firma del método es la siguiente:

public void drawRangoMove(int x, int y, int desplazamiento, World.type tipo, World.option opcion)

Los parámetros “x” e “y” sirven para marcar la posición del pet solicitante. “desplazamiento” sirve para saber cuánto se puede desplazar el pet en cuestión. “tipo” es un dato enumerado definido en la clase World que nos servirá para saber el tipo del pet. “opción” otro parámetro enumerado, nos indicará si lo que desea el jugador es atacar o mover.

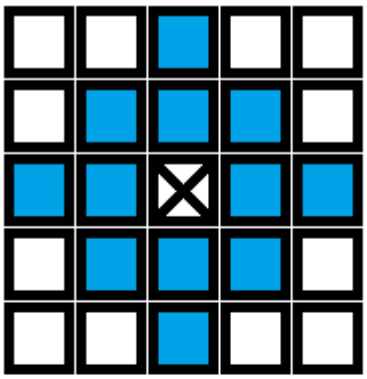
Ahora veremos cómo construimos el rango con todos estos datos.

Lo primero que debemos saber es cuanto se puede mover el pet. Esto viene definido por la estadística displacement que poseen todos los tipos de P.U Pets. Las estadísticas sirven para que el jugador pueda comparar unidades y saber cuáles son más útiles en su estrategia. Pero estas estadísticas hay que transformarlas en datos reales con los que trabajar. En el caso del desplazamiento, su valor real es el de la estadística dividida entre 10. Esto marcará el número de casillas que se puede mover. Por ejemplo, si diplacement es 30, el pet se podría mover 3 casillas en todas direcciones a partir de su posición actual.

La misma lógica es aplicable al ataque, teniendo en cuenta que el parámetro desplazamiento ha de ser informado con la estadística attack_range.

Como se ha comentado anteriormente, el rango englobará las casillas contiguas a la posición actual en todas direcciones. Esto se formaría teniendo en cuenta que la suma de las diferencias entre las coordenadas de la posición inicial y las coordenadas de la casilla tratada no sea mayor que el parámetro desplazamiento

Ilustración 18: Ejemplo de formación del rango de movimiento



posición actual (5,5)
displazament=20
desplazamiento=20/10=2

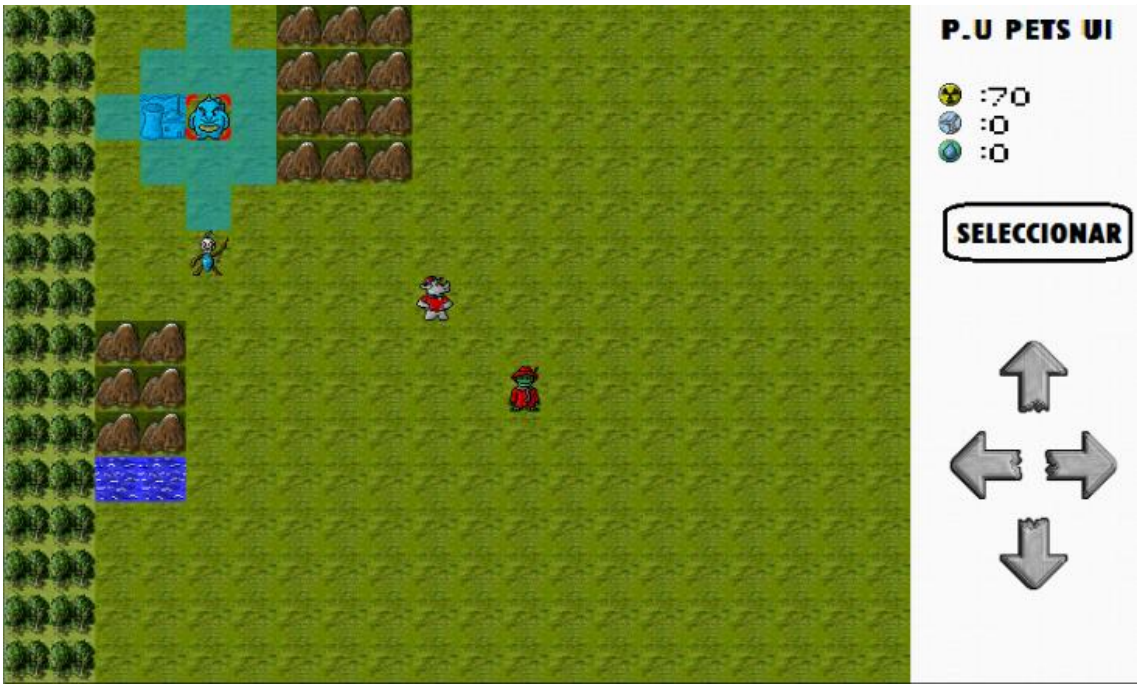
Rango
(5,3)
(4,4)(5,4)(6,4)
(3,5)(4,5)X(6,5)(7,5)
(4,6)(5,6)(6,6)
(5,7)

(5,3) --> Diferencias |5-5|=0 + |3-5|= 2 --> 2<=desplazamiento
(4,4) --> Diferencias |4-5|=1 + |4-5|= 1 --> 2<=desplazamiento
(5,4) --> Diferencias |5-5|=0 + |4-5|= 1 --> 1<=desplazamiento
(6,4) --> Diferencias |6-5|=1 + |4-5|= 1 --> 2<=desplazamiento
(3,5) --> Diferencias |3-5|=2 + |5-5|= 0 --> 2<=desplazamiento
(4,5) --> Diferencias |4-5|=1 + |5-5|= 0 --> 1<=desplazamiento
(6,5) --> Diferencias |6-5|=1 + |5-5|= 0 --> 1<=desplazamiento
(7,5) --> Diferencias |7-5|=2 + |5-5|= 0 --> 2<=desplazamiento
(4,6) --> Diferencias |4-5|=1 + |6-5|= 1 --> 2<=desplazamiento
(5,6) --> Diferencias |5-5|=0 + |6-5|= 1 --> 1<=desplazamiento
(6,6) --> Diferencias |6-5|=1 + |6-5|= 1 --> 2<=desplazamiento
(5,7) --> Diferencias |5-5|=0 + |7-5|= 2 --> 2<=desplazamiento

Luego, solo hay que construir el rango recorriendo las posiciones contiguas a la posición actual del pet y comprobar si cumplen los requisitos para formar parte del rango de movimiento. Para ello, en el caso del movimiento, hay que tener en cuenta el tipo de terreno almacenado en la casilla de la primera dimensión de la matriz. Por ejemplo, podría darse el caso de una unidad del tipo Schlach situado junto a unas montañas.

Esta unidad no puede pasar por las montañas, así que a pesar de que varias de estas casillas, si tuviéramos en cuenta solo la distancia, entrarían dentro del rango de movimiento, en este caso no es así puesto que el tipo de pet es incapaz de recorrer dicha zona.

Ilustración 19: Ejemplo de imposibilidad de movimiento por determinadas casillas



El caso de ataque, es distinto. No se tiene en cuenta el tipo de terreno. Tampoco se tiene en cuenta el tipo de pet al que se ataca. Una unidad puede atacar a otra que está dentro de su rango, pero hay que tener en cuenta que hay determinados tipos que no pueden alcanzar a otros.

7.3 IMPLEMENTACIÓN DE LA IA

Para el modo un jugador, será necesario facilitarle al usuario un rival. De esto se encarga las llamadas IA o inteligencia Artificial. Se trata de un algoritmo que simula las acciones de otro jugador. En el caso del presente proyecto, se trata de un algoritmo que se vale de los métodos del código que se utilizan para interacciones con un usuario real.

Este algoritmo tiene lugar en la clase “World”, pero se vale de una instancia de la clase “RobbieIA” que contiene algunos métodos de análisis;

select_plant_moved sirve para elegir una planta energética como objetivo. Busca la más cercana al pet que se está tratando.

“comprobar_enemigos” sirve para elegir un pet enemigo como objetivo. Comprueba si hay algún enemigo al que pueda atacar (si el pet del usuario es de un tipo al que no puede atacar con el que está atacando, lo descartará) a menos de 6 casillas de la posición actual (en ambos ejes) de pet que se está tratando.

"comprobar_enemigos_especiales" cuenta el número de unidades no terrestre (es decir, aéreas o acuáticas) que tiene el usuario.

"comprobar_IA_especiales" cuenta el número de unidades no terrestre (es decir, aéreas o acuáticas) que tiene a su disposición la IA.

"control_creacion_especiales" se vale de los dos anteriores métodos para controlar si es necesario que la IA cree unidades capaces de contrarrestar a las especiales del enemigo. Por ejemplo, si el usuario tiene 3 unidades aéreas y la IA solo 1, este método le indicará que necesitaría una unidad capaz de contrarrestarla. Crearía en ese caso una unidad aérea o un Tiraespinas.

El algoritmo va tratando una a una las unidades de su bando, calculando sus movimientos y ataques.

En la clase World, también dispone de algunos métodos específicos para su uso exclusivo como son comprobar_ataque y comprobar_ataque_primero, que se usan para comprobar si la unidad que está tratando la IA tiene posibilidad de atacar a un pet enemigo. Uno se utiliza para comprobarlo antes de que la unidad haya movido y el otro para comprobarlo después.

La IA, va también va comprobando cuantas y cuáles de las centrales energéticas contenidas en el array_nuclear que es donde están contenidas todas las centrales energéticas. En cada turno, de ser posible, creará una unidad por cada central de su bando. La creación depende de la radioactividad, energía eólica y energía hidráulica que tenga disponible, así como de las unidades especiales de las que disponga el usuario, como hemos explicado anteriormente.

Con todo esto, la IA supondrá realizará sus movimientos y ataque para dar la sensación de estar jugando contra "alguien".

8 PROTOTIPADO

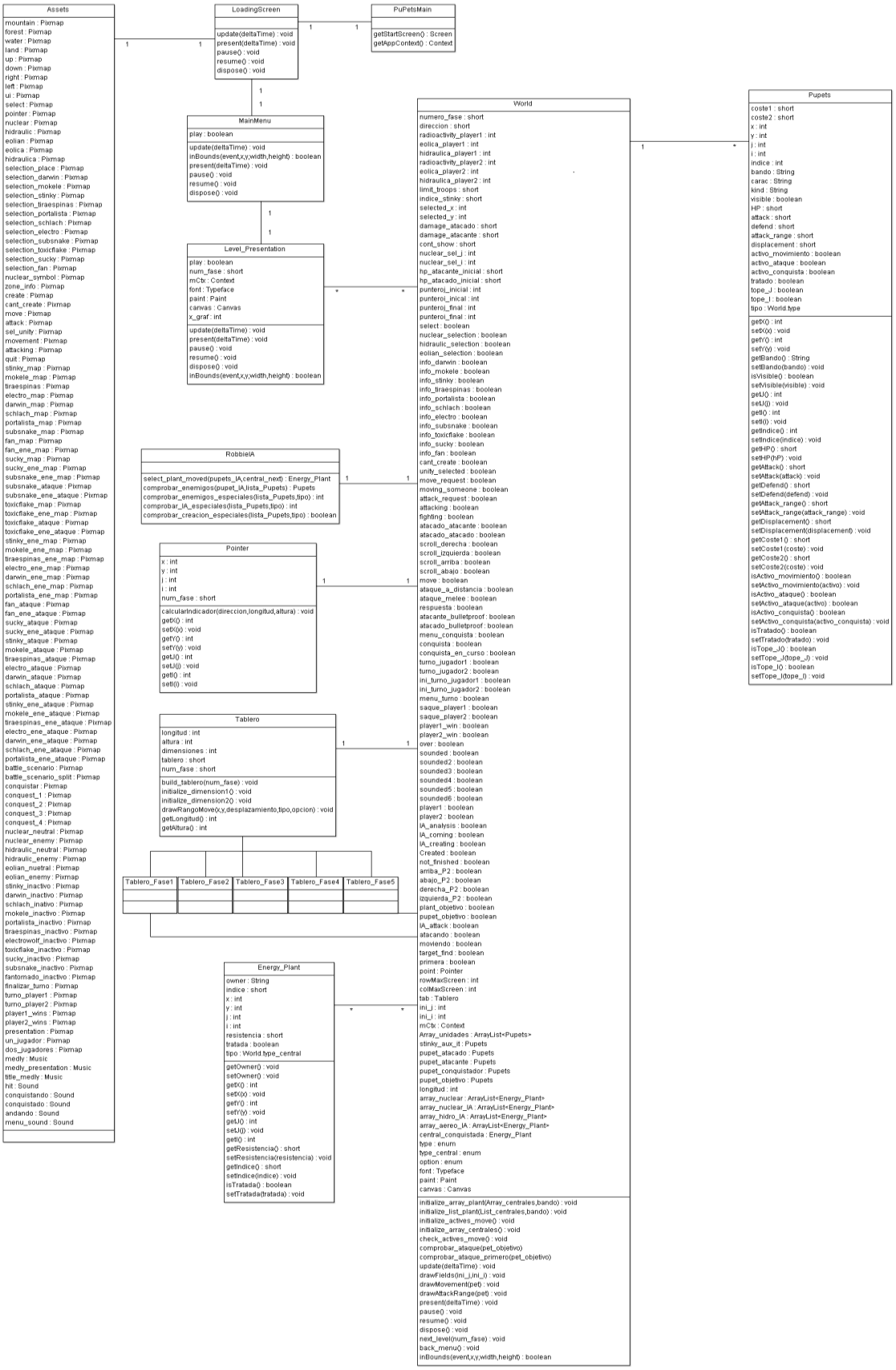
Una vez tuvimos la parte básica del desarrollo, se sometió a pruebas el prototipo con el propósito de encontrar deficiencias en el diseño inicial.

Los primero que se observó, es que las estadísticas de las unidades no estaban bien compensadas. Tras algunas pruebas, fueron reajustadas buscando el equilibrio.

También se producían desajustes como producto de los ataques especiales. En una de las partidas, un jugador subió las estadísticas de una unidad "Mokele Embembe" consiguiendo ganar la partida fácilmente usando solamente este pet. Se toma la drástica decisión de eliminar los movimientos especiales, para evitar estos casos.

Tampoco se ha incluido la unidad "Buss" en la versión final, en este caso por falta de tiempo.

8.1 DIAGRAMA DE CLASES FINAL



La clase “World” es la encargada de gran parte del trabajo. Se nutre de instancias de las otras clases de juego como por ejemplo “Pupets” o “Energy_Plant” para presentar en pantalla el hilo principal.

Tenemos una clase “Tablero” que es la encargada de las reglas del tablero para dibujar el rango de movimiento y ataque de los pets. Tienen instancias de “Tablero_FaseX” para construir el escenario correspondiente a la fase actual.

“RobbieIA” se encarga de los métodos de análisis para la inteligencia artificial.

“Level_presentantion” sirve de pantalla de carga de parámetros, como enemigos o centrales energéticas, necesarios en las instancias de “World” que forman las distintas pantallas.

“Assets” es la pantalla donde se definen todos los gráficos, sonidos y música almacenados en la carpeta del mismo nombre y que cargaremos en “LoadingScreen”.

9 MANUAL

9.1 INTRODUCCIÓN

¡Bienvenido al mundo de P.U Pets!

Enfréntate a tus enemigos a través de tu móvil Android y bórralos del mapa en una encarnizada lucha por la supervivencia de tu bando.

9.2 HISTORIA

Año 20XX* D.C. Con el fin de los combustibles fósiles comienza la irremediable búsqueda de una alternativa real por parte de la comunidad científica. Proliferan las centrales energéticas que utilizan energías renovables como la eólica o la hidráulica. Sin embargo, para sorpresa de todos, también comienza un crecimiento exponencial del número de centrales nucleares con motivo del descubrimiento de un proceso para el reciclado de los residuos. El descubridor fue un científico Uzbeco llamado Pavel Uslamovich por lo que el proceso sería conocido como “proceso de reciclado P.U”.

Paralelamente, la industria juguetera, en clara progresión descendente durante los últimos años, buscaba un producto que consiguiera interesar a todos y suponer una verdadera revolución en el campo del entretenimiento. Surge el planteamiento de crear seres orgánicos que sirvan como mascotas, pero sin sus inconvenientes. Este proyecto, fue denominado con el nombre en clave “Pets”.

Todos los intentos para llevarlo a cabo fracasaron, hasta que un ingeniero, desesperado en su intento, usó residuo radiactivo como materia prima siguiendo un proceso similar al P.U. Fue entonces cuando dio con la clave; ¡podía crear las mascotas a través del proceso de reciclado! Esto supuso una autentica revolución. “Un ejercicio de sinergia sin precedentes” fue denominado.

Pronto fueron presentados en sociedad los “P.U Pets”, mascotas perfectas sin ningún tipo de necesidad.

Los beneficios de la industria juguetera se dispararon y se contaban por miles de millones a la vez que se reciclaba todo el residuo nuclear que se creaba. Sin embargo, esto no fue considerado suficiente. Para la creación de los P.U Pets era necesario un costoso proceso para el que se precisaba un gran número de mano de obra de alta cualificación. Por eso se comenzó el proyecto “Mother”, que perseguía la elaboración de un tipo de P.U Pet especial, capaz de crear por el mismo las unidades precisadas para su venta.

No se tardo demasiado en conseguirlo. Se crearon dos unidades Mother que abastecían todo el suministro necesario mientras la energía nuclear proliferaba hasta niveles inauditos hasta la fecha. Fue entonces cuando la naturaleza más volátil de este tipo de energía hizo acto de presencia. Se desencadeno una serie de explosiones en cadena, que desembocaron en la mayor catástrofe de la historia de la humanidad.

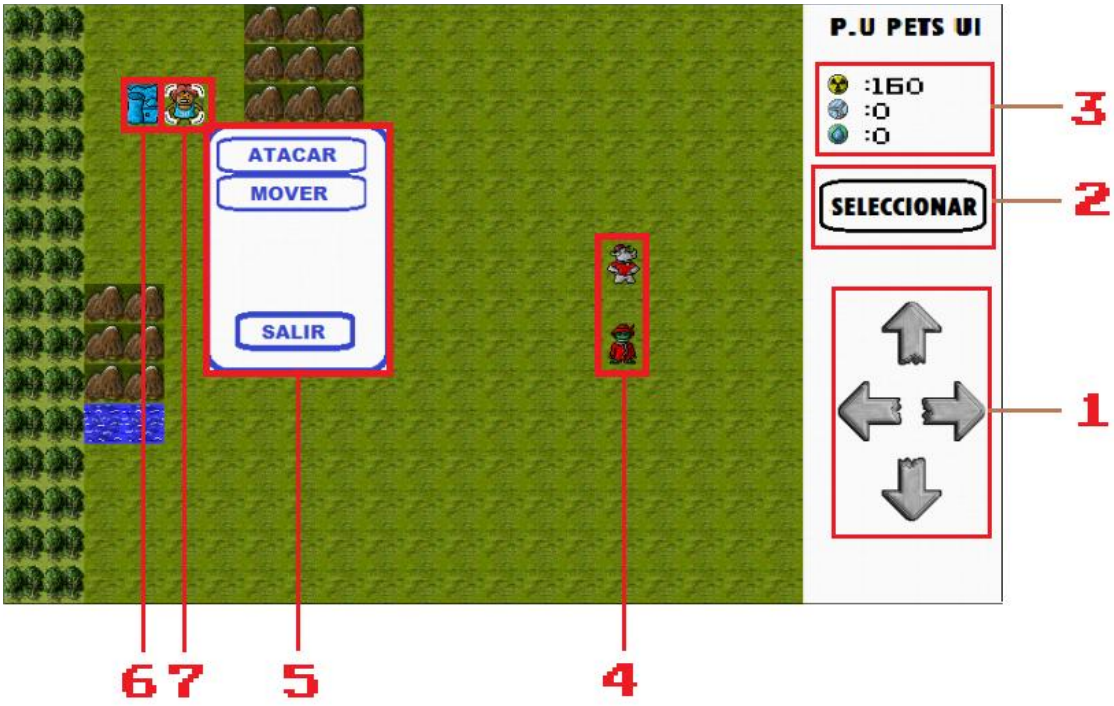
Pocos fueron los supervivientes, que iban siendo diezmados por el veneno radiactivo que lo poblaba absolutamente todo. Sin embargo, esto supuso unas condiciones idóneas para las Mother, que comenzaron una lucha por la posesión de las centrales enérgicas con el propósito de hacer proliferar su prole…

9.3 COMO JUGAR

P.U Pets es un juego de estrategia por turnos. Esto significa que cada jugador debe esperar a su turno correspondiente para realizar sus jugadas.

La pantalla de juego es similar a la siguiente:

Ilustración 20: Explicación de los elementos de la pantalla.



1 – FLECHAS DE MOVIMIENTO

Para mover el cursor de juego[8] por el escenario, el jugador debe usar estas fechas. Es necesario para interactuar con los distintos elementos. El jugador debe pulsar la flecha correspondiente a la dirección a la que quiere desplazar el cursor.

2 – BOTÓN DE SELECCIÓN

Sirve para acceder al menú contextual[5]. Selecciona el elemento con el que desea interactuar.

3 – INDICADORES DE ENERGÍA

Para generar las unidades P.U Pets[7] de tu bando, hay que consumir un precio en energía. El valor de este precio depende de la unidad que se quiere construir.

Los distintos indicadores enérgicos de tu bando aumentan en 10 en cada turno por cada central energetica[6] que sirva a tu facción.

Los distintos indicadores marcan lo siguiente:

: Indica la energía nuclear disponible.

: Indica la energía eólica disponible.

: Indica la energía hidráulica disponible.

4 – UNIDADES P.U PETS ENEMIGAS

Las unidades P.U Pets y centrales marcadas en rojo, pertenecen al bando enemigo. Son las que deben ser destruidas/conquistadas para alcanzar la victoria.

5 – MENÚ CONTEXTUAL

Cuando seleccionas algo con lo que puedes interactuar, surgirá este tipo de menú. Será diferente según lo seleccionado. En caso de seleccionar una zona baldía, sin nada aparte de terreno, surgirá el menú utilizado para finalizar el turno. Para seleccionar algunas de las opciones presentadas, basta con tocarla.

Los distintos tipos de menú contextual, son los siguientes:

Al seleccionar una central energética de tu bando → **Menú de creación de unidades:**

Ilustración 21: Ejemplo menú de creación de unidades

Al seleccionar en el menú anterior una unidad a crear → **Menú de estadísticas:**

Indica al jugador las estadísticas de la unidad P.U Pet a crear, dependiendo de su tipo:

Ilustración 22: Ejemplo de pantalla de estadisticas

En el caso de no poseer suficientes recursos energéticos para crearla o de encontrarse ocupada la zona de creación[9], el botón de creación aparecerá de la siguiente forma:

Ilustración 23: Ejemplo de botón crear inhabilitada

Como indicativo de la incapacidad para crear la unidad.

Al seleccionar una unidad P.U Pet de tu bando → **Menú de unidad:**

Sirve para realizar acciones con la unidad seleccionada. Al pulsar sobre las opciones ATACAR/MOVER, se presenta el rango de movimiento10/rango de ataque11 de la unidad y el usuario tendrá que desplazar el cursor hacía la casilla/enemigo sobre la que se desea realizar la acción.

Ilustración 24: Ejemplo de menú de unidad

Al seleccionar una casilla vacía (que presenta solo el terreno) → **Menú de finalización de turno:**

Situé el cursor sobre una casilla sin centrales o unidades P.U Pets para acceder a este menú. Se usa cuando el jugador ha terminado de hacer su jugada y tiene que pasar turno.

Ilustración 25: Ejemplo de menú de finalización de turno

Al seleccionar una unidad P.U Pet de tu bando que esta posicionado en una central energética que no es de tu bando→ **Menú de unidad conquista:**

Cuando la unidad seleccionada está situada en una central energética que no es de tu bando, el menú de unidad cambiará y dará la posibilidad de comenzar la conquista.

Ilustración 26: Ejemplo de menú de conquista

6 – CENTRALES ENERGETICAS

Proporcionan energía cada turno y permiten la creación de P.U Pets de tu bando7. En caso de ser enemigas (vienen definidas en color rojo), cumplen la misma función para la facción contraría, por lo que conviene arrebatárselas a través de la conquista. También existen centrales sin bando, también llamadas neutrales (definidas en color gris) , que pasaran a ser de la primera facción que las conquiste.

Las centrales energéticas pueden ser de varios tipos:

Nucleares: Proporcionan energía nuclear y permiten la creación de unidades terrestres.

Hidráulicas: Proporcionan energía hidráulica y permiten la creación de unidades acuáticas.

Eólicas: Proporcionan energía eólica y permiten la creación de unidades aéreas.

7 – UNIDADES P.U PETS DE TU BANDO

Son las unidades pertenecientes a tu facción. Tienen un color azulado y puedes seleccionarlas para interactuar con ellas a través del menú contextual[5]. Se crean a partir de una central energética[6] de tu bando.

8 – CURSOR DE JUEGO

Sirve al jugador para interactuar con los elementos de juego y se desplaza a través de las flechas de movimiento[1].

9 – ZONA DE CREACIÓN

Es la casilla que está justo a la derecha de las centrales. En el caso de estar ya ocupadas con una unidad P.U Pet, se impediría la creación de nuevas unidades.

10- RANGO DE MOVIMIENTO

Cuando el jugador desea desplazar una de sus unidades y selecciona dicha opción, se le presentará su correspondiente rango de movimiento. Para ello, se marcarán las casillas a las que se pueda desplazar en azul. Para salir del modo movimiento, bastará con desplazar el cursor de juego[8] fuera del rango de movimiento.

Ilustración 27: Ejemplo de Rango de movimiento

11- RANGO DE ATAQUE

Cuando el jugador desea atacar a una de las unidades enemigas debe hacerlo a través de unos de sus P.U Pets. Para ello debe seleccionarlo y usar la opción ATACAR. Se le presentará su correspondiente rango de ataque. Para ello, se marcarán las casillas englobadas en este rango

en rojo. Si la unidad enemiga, está en una de estas casillas, tendrá la posibilidad de atacarla seleccionándola con el cursor de juego[8]. Si desea salir del modo ataque, bastará con desplazar el cursor de juego[8] fuera del rango de ataque.

Ilustración 28: Ejemplo de rango de ataque

9.4 COMO GANAR

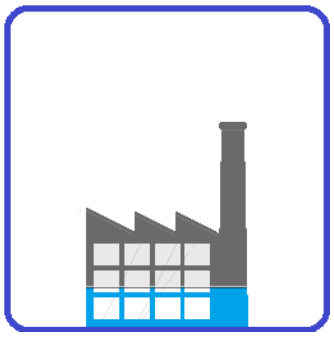
Gana la partida, el bando que conquiste todas las centrales energéticas disponibles por el mapa o que acabé con absolutamente todas las unidades del bando contrario.

9.5 COMO CONQUISTAR UNA CENTRAL ENERGETICA

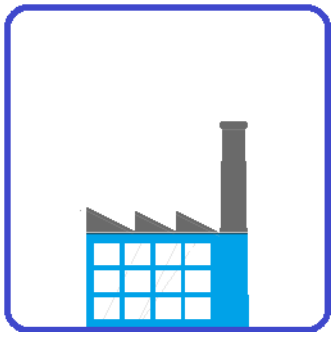
Para conquistar una central energética, debes posicionar una de tus unidades P.U Pets sobre una central energética que no sea de tu bando y elegir la opción del menú contextual CONQUISTAR.

Las centrales energéticas tienen resistencias, por lo que serán precisos varios turnos para lograr la conquista completa. El número de turnos que sea precisos dependen de los HP que tenga la unidad conquistadora, a más HP, más resistencia le resta cada turno.

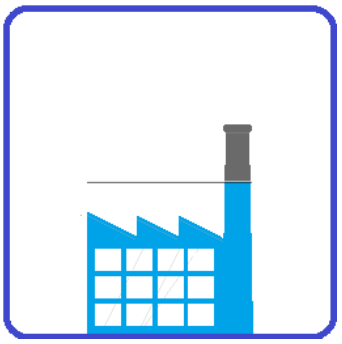
Los indicativos del nivel de conquista, son los siguientes:



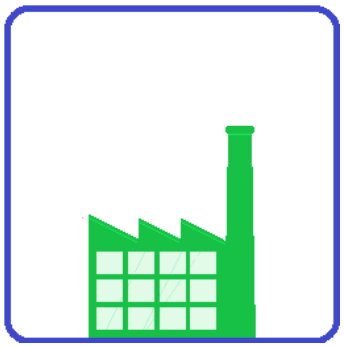
Nivel de conquista bajo



Nivel de conquista medio



Nivel de conquista alto



Central conquistada

9.6 COMO ELIMINAR LAS UNIDADES P.U PETS ENEMIGAS

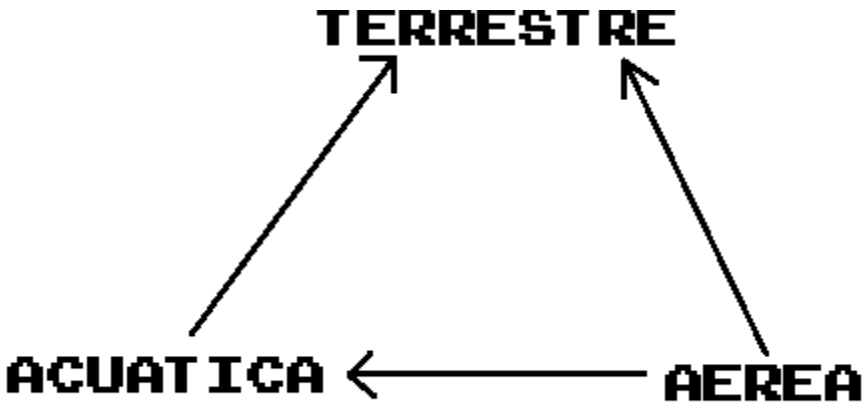
Simplemente tienes que atacarlas hasta que sus HP llegue a 0.

9.7 ELEMENTO DE LAS UNIDADES P.U PETS

Las unidades P.U Pets, como las centrales, pueden ser de tres elementos; terrestres, aéreos y acuáticos. Esto tiene repercusión en las batallas puesto que algunos tipos son inmunes a otros.

La correspondencia de inmunidad sería la siguiente:

Ilustración 29: Posibilidad de ataque entre tipos



Sin embargo, existen algunos tipos de unidad P.U Pets especiales, como son ElectroWolf y Tiraespinas que a pesar de ser terrestres, puede atacar a unidades acuáticas y aéreas respectivamente.

9.8 CASOS ESPECIALES DE MOVIMIENTO

Existen algunas unidades P.U Pets que no se pueden mover por determinado tipo de terreno. Por ejemplo, Stinky Sting no puede moverse por el agua, sin embargo ToxicFlake solo puede moverse por ese elemento. En ese caso, las casillas no formarán parte del rango de movimiento.

9.9 RESPUESTA A LOS ATAQUES

Algunas unidades responden inmediatamente a los ataques enemigos. Eso depende del tipo de ataque que utilicen. Estos tipos son los siguientes; melee, mixto y largo alcance. Los dos primeros son capaces de responder. Para saber qué tipo tiene una unidad P.U Pet, solo hay que observar su rango de ataque. Si solo pueden atacar a las casillas contiguas, son de tipo melee, si además pueden atacar a más casillas, serán mixtos y si no pueden atacar a las casillas contiguas pero puede hacerlo a larga distancias, serán de largo alcance.

9.10 TIPOS DE UNIDADES

STINKY STING

Coste: 20 radioactividad

Ataque: 100

Defensa: 10

HP: 30

Desplazamiento: 30

Rango Ataque: 30

DARWIN

Coste: 20 radioactividad

Ataque: 70

Defensa: 10

HP: 30

Desplazamiento: 20

Rango Ataque: 50

MOKELE EMBEMBE

Coste: 100 radioactividad

Ataque: 130

Defensa: 10

HP: 30

Desplazamiento: 30

Rango Ataque: 10

Características Especiales: No puede moverse por montañas.

SCHLACH

Coste: 100 radioactividad

Ataque: 70

Defensa: 80

HP: 30

Desplazamiento: 20

Rango Ataque: 50

Características Especiales: No puede moverse por montañas.

ELECTROWOLF

Coste: 110 radioactividad

Ataque: 90

Defensa: 40

HP: 30

Desplazamiento: 20

Rango Ataque: 20

Características Especiales: Puede atacar unidades acuáticas.

TIRAESPINAS

Coste: 110 radioactividad

Ataque: 100

Defensa: 40

HP: 30

Desplazamiento: 20

Rango Ataque: 20

Características Especiales: Puede atacar unidades aéreas.

PORTALISTA

Coste: 50 radioactividad

Ataque: 70

Defensa: 30

HP: 15

Desplazamiento: 80

Rango Ataque: 30

SUBSNAKE

Coste: 30 radioactividad + 20 hidráulica

Ataque: 30

Defensa: 100

HP: 20

Desplazamiento: 20

Rango Ataque: 20

Caracteristicas Especiales: Solo se mueve por el agua y solo acierta sus ataques en unidades acuaticas.

TOXICFLAKE

Coste: 60 radioactividad + 30 hidráulica

Ataque: 90

Defensa: 90

HP: 30

Desplazamiento: 20

Rango Ataque: 30

Caracteristicas Especiales: Solo se mueve por el agua.

FAN TORNADO

Coste: 30 radioactividad + 20 hidráulica

Ataque: 100

Defensa: 20

HP: 10

Desplazamiento: 40

Rango Ataque: 10

Caracteristicas Especiales: Solo acierta sus ataques en unidades aereas.

SUCKY

Coste: 60 radioactividad + 30 hidráulica

Ataque: 20

Defensa: 90

HP: 30

Desplazamiento: 40

Rango Ataque: 10

10 TESTING

Como se ha indicado previamente, una labor de testing más amplia sería necesaria para pulir el proyecto antes de su salida al mercado.

INFORME DE ERRORES:

	Descripción	Causas	Solución
Error 1	Imposibilidad de interactuar con los elementos (unidades, centrales) después de usar el scroll.	Descoordinación entre las coordenadas del puntero y del mapa al usar el scroll.	Cambio en el algoritmo del scroll, sumando 1 a los parámetros ini_j/ini_i después de haberse solicitado el scroll y no antes.
Error 2	Unidades comandadas por la IA, se quedan inmóviles tras encontrar un obstáculo en su camino.	Falta de un algoritmo de búsqueda de caminos.	Desarrollo de un rudimentario método de pathfinding en el que si las unidades encuentran un obstáculo en el eje y, intentan avanzar en el eje x y viceversa.
Error 3	Unidades de la IA atacan dos veces en el mismo turno.	Al hacer dos comprobaciones para atacar, sucedía que en ocasiones,	Comprobar el estado de “activo ataque” de las unidades. De esta manera, una vez

		atacaban dos veces.	han atacado, ya no podían atacar de nuevo en ese mismo turno.
Error 4	Al acabar con todas las unidades enemigas, la partida no acaba y la última unidad vuelve a aparecer y sigue atacando, eliminando una unidad del usuario por turno.	Se eliminaban las unidades del arraylist por índice. Esto provocaba que tras borrar una, los índices se reorganizaran. Esto producía un desajuste entre las unidades.	Borrar del arraylist según la unidad completa y no según índice (usando el método de las arraylist .remove(Object)
Error 5	Dobles desplazamientos con cada pulsación en las flechas de movimiento.	El listener de los eventos táctiles, notificaba una pulsación cuando el dedo pulsaba y otro cuando el dedo dejaba de pulsar.	Se incluye una comprobación extra, para ver si es pulsación down o up. Se tiene en cuenta solo las down. Esto provoca que en dispositivos móviles el funcionamiento sea el esperado. Sin embargo, al probar en AVD´s, las pulsaciones no son siempre tratadas.

11 CONCLUSIÓN

Mi primer contacto con el desarrollo de los videojuegos ha servido para reafirmarme en mi vocación. El desarrollo ha sido muy complicado, siendo especialmente difícil ser capaz de aprender Android, desarrollar un juego y testearlo en tan solo cuatro meses. Sin embargo, he disfrutado mucho y planeo mejorar el actual proyecto y hacer cambios significativos en él, además de continuar mi formación con el objetivo de dedicarme a esta industria profesionalmente.

12 MUSICA Y SONIDOS

Música:

Megaman - Capcom - Nintendo Entertaiment System

Silver Surfer - Software Creations - Nintendo Entertaiment System

Blaster Master – Sun Soft - Nintendo Entertaiment System

Sonidos:

http://www.soundsnap.com/tags/8_bit

http://www.bfxr.net/

13 BIBLIOGRAFIA

Zechner M.; *Desarrollo de juegos ANDROID*. Titulo de la obra original: Beginning Andriod Games; Madrid: Ediciones Anaya Multimedia (grupo Anaya S.A),2012.

ISBN: 978-84-415-3035-5