Informe proyecto final
Crónicas del Contagio - videojuego

Michell Dayana Gaitán Gutiérrez
Manuel José Montoya Arboleda

2025-2
UdeA
Informática II
Profesor Aníbal José Guerra Soler

Introducción

Este proyecto surge como parte del reto final del curso, donde debíamos crear un videojuego en C++ usando los recursos de Qt y aplicando la programación orientada a objetos, memoria dinámica, modelos físicos y diseño de interfaces.
El objetivo fue construir un juego de tres niveles, cada uno inspirado en un momento histórico diferente, buscando que fuera llamativo, entendible y divertido de jugar, incluyendo sistemas físicos, elementos inteligentes y escalabilidad en la dificultad.

A lo largo del semestre el proyecto se desarrolló por etapas: primero se definió el contexto y las dinámicas generales del juego (Momento I), luego se diseñó la estructura lógica y el diagrama de clases (Momento II), y finalmente se implementó el videojuego completo (Momento III).

En este informe se explica todo el proceso: qué se buscó lograr, cómo se diseñó la arquitectura del juego, qué decisiones técnicas se tomaron y cómo fue progresando el proyecto hasta llegar a la versión final. También se describen las físicas usadas, la forma en que se integró el agente inteligente, y los recursos de Qt usados.

En los cursos de programación suele abordarse de manera independiente conceptos como la memoria dinámica, las interfaces graficas o la POO, por ende, el principal objetivo del proyecto es integrar el conocimiento adquirido durante todo el curso con una aplicación interactiva que utiliza todos los elementos mencionados y nos permite desarrollar un producto tangible que involucra problemas reales de diseño y estructuración.
Además, desarrollamos el juego como una forma dinámica de interactuar con la historia y aprender sobre ella, creando una plataforma accesible para cualquier público interesado.

El problema central que aborda el juego es representar, mediante mecánicas interactivas, las diferencias entre tres momentos históricos donde la medicina enfrentó desafíos distintos.

Planteamiento del problema

A través de nuestro juego nos propusimos representar la evolución de la ciencia y tecnología aplicadas a la medicina, lo que se ha visto históricamente reflejado en la manera que se han desarrollado las pandemias que han azotado a la humanidad. Seleccionamos tres pandemias históricas claves en orden cronológico: la peste negra, la primera pandemia del colera y la pandemia del COVID-19. Las anteriores funcionan perfectamente como demostración de los avances en la medicina por la amplia distancia temporal que los separa y los contextos sociales bajo los cuales se desarrollaron.

Nuestro primer nivel se ambienta durante la peste negra, tiempo en el cual el conocimiento medico era muy limitado y los tratamientos en su mayoría empíricos y con efectos reales limitados, dado al poco conocimiento que se tenia sobre el funcionamiento del cuerpo humano. En aquella época no se entendían las causas de las enfermedades, los medios de trasmisión ni los tratamientos para curarlas; la enfermedad era un castigo divino o un mal aire que se esparcía con el viento y era tratado con vapores de hierbas y plantas.
Por lo anterior, en el primer nivel el jugador controla a un ciudadano europeo que vive durante la peste, cuyo objetivo es escapar de una multitud de personas infectadas por la peste que corren desorientadas por las calles sin ser infectado por ellas, mientras recoge hierbas y frutos medicinales que otras personas han dejado caer. Adicionalmente el jugador debe escapar de cierta persona infectada que tratara de robarle las hierbas que el jugador ha recolectado.

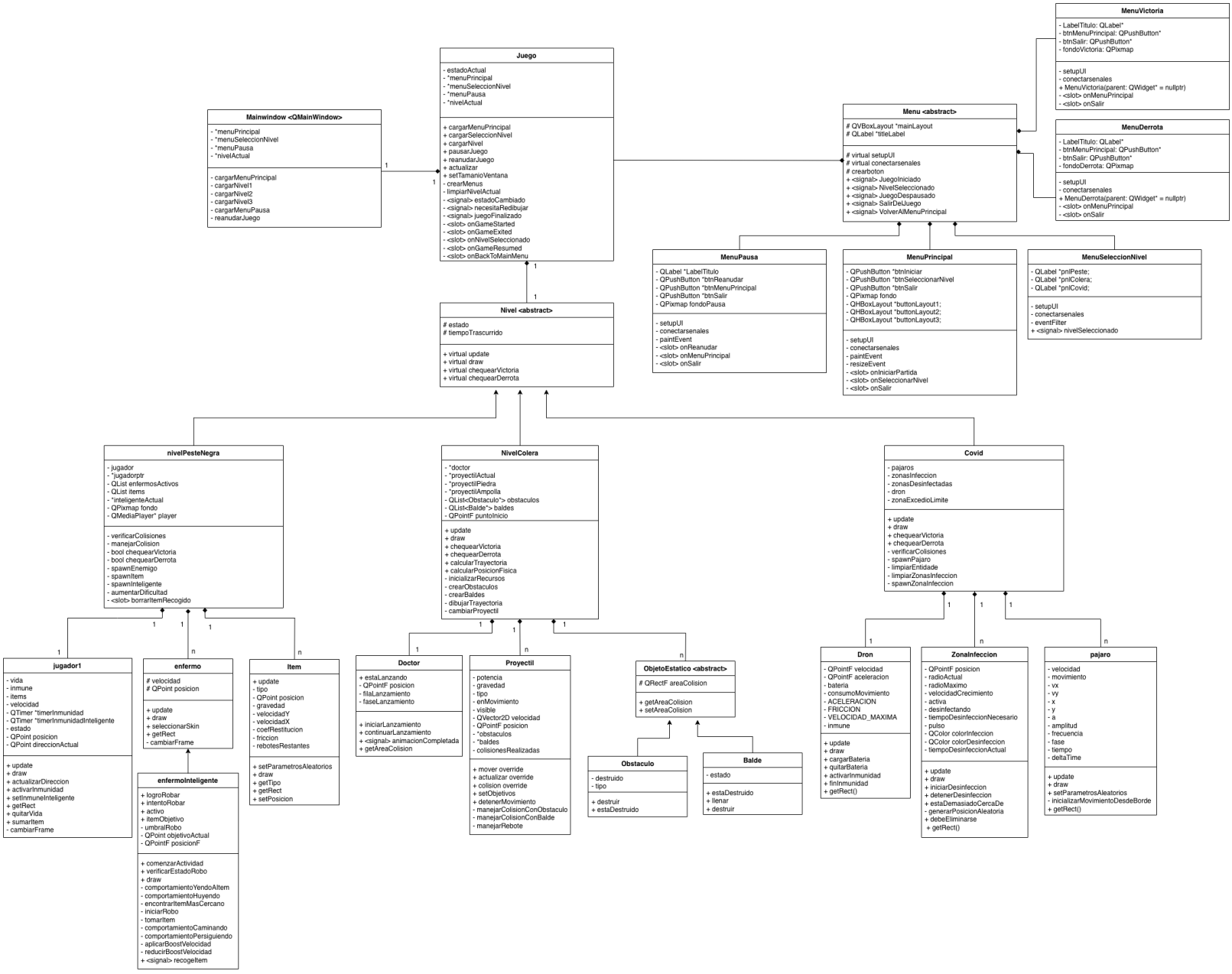
El segundo nivel tiene lugar durante la primera pandemia del colera. Para este momento de la historia ya se tenía un conocimiento médico del cuerpo humano y de las enfermedades más desarrollado, lo que permitió identificar las causas de la propagación del colera y empezar tratamientos preventivos; se descubrió que el virus habitaba en aguas contaminadas por el pobre manejo de residuos en la época y era trasmitido por el consumo de dicha aguda.
En este nivel el jugador encarna a un médico de la época, que ha desarrollado un desinfectante para las aguas contaminadas con el virus. El jugador debe lanzar las ampollas con desinfectante hacia los baldes de agua contaminada de un pueblo cercano, teniendo cuidado de no desperdiciar el desinfectante.

El tercer nivel de nuestro juego se desarrolla durante la reciente pandemia del COVID-19, durante la cual se evidencia una ventaja abismal en comparación con las dos anteriores, en términos de tratamientos y prevención de la enfermedad. Durante la pandemia del COVID, la tecnología y estudios médicos permitieron hallar de una manera fidedigna los medios de trasmisión de la enfermedad, estudiar el virus, ofrecer tratamiento a las personas infectadas y medidas de prevención para las personas en riesgo de contraer la enfermedad, e inclusive desarrollar una vacuna para combatir al virus.
En este nivel el jugador controla a un dron que sobrevuela una ciudad en la que tiene que entregar vacunas y medicamentos para las personas en cuarentena. El jugador debe estar atento a las zonas de infección marcadas en el mapa para atenderlas, y esquivar a los pájaros que vuelan sobre la ciudad y hacen daño al dron cuando chocan con él.

En cada nivel el jugador puede moverse y controlar elementos mediante teclado y mouse, ejecutar acciones, interactuar con los objetos e influir en la física de la escena. Todos los

niveles incluyen al menos dos sistemas físicos distintos, se registran y procesan colisiones constantemente y el primer nivel incorpora el agente inteligente requerido.
Además, cada nivel cuenta con música y efectos de sonido, así como un sistema completo de menús para facilitar la interacción con el usuario.
El alcance del juego se centra en la jugabilidad en solitario (no multijugador) y la ejecución exclusiva en sistemas operativos Windows.

Diseño y arquitectura del sistema



Para nuestro programa utilizamos la clase mainWindow como el motor para el entorno gráfico, encargada de ajustar los tamaños de las ventanas y gestionar las invocaciones para mostrar los menús.

En las clases correspondientes a los niveles del juego utilizamos los métodos update y draw para generar las verificaciones de interacciones en el juego y redibujar en pantalla a 60fps, mediante la invocación recursiva de update con un QTimer en la clase juego y la invocación

a draw en mainWindow cuando recibe la señal necesitaRedibujar emitida por update de juego.

En general, las clases que necesitaran verificar colisiones heredan de las clases QObject o QGraphicsPixmapItem y utilizan el método getRect, que retornaQRect, para obtener su hitbox y verificar colisiones. Las posiciones de las entidades colisionables que son retornadas con getRect se guardan en atributos de tipo QPoint.

Desarrollo e implementacion:

Juego

La clase contiene a las clases de menús y a la clase nivel. Controla el estado y los resultados de la partida y a partir de ello invoca métodos de carga de los menús, carga de los niveles y reanudar o pausar la partida, además de entregarle el manejo de imputs al nivel actual y ajustar el tamaño de la ventana.
Contiene un QTimer timerJuego que funciona a 60fps e invoca constantemente al método actualizar, que a su vez invoca al método update del nivel actual y genera una señal necesitaRedibujar, que es recibida por mainWindow para mostrar las animaciones en pantalla invocando al método draw del nivel actual.

menu

Clase abstracta que hereda de QWidget. Contiene un QVBoxLayout layout vertical centrado para posicionar los botones de los menús, un QLabel para el titulo y un metodo QPushButton para crear los botones. Emite señales de inicio de la partida, nivel seleccionado, volver al menú principal, entre otras, permitiendo que cualquier menú se comunique con la clase juego.

menuPrincipal, menuPausa, menuSeleccionNivel

Heredan de la clase menu y funcionan como widgets independientes utilizados por mainWindow que implementan su propia interfaz y botones con setupUI y conectarSenales. Se comunican con la clase juego a travez de señales. mainWindow los ancla a la ventana y segun el estado del juego utiliza show o hide. Son manejados por mainWindow ya que es quien tiene acceso al paintEvent de la ventana.

nivel

clase abstracta que hereda de QObject. Contiene los metodos virtuales update, draw, manejo de imputs y verificaciones de victoria y derrota. También guarda el tiempo de duración de la partida y el tamaño de la ventana

nivelPesteNegra

Hereda de nivel. Contiene un objeto de la clase jugador1, listas con apuntadores a objetos de clase item y enfermo, y un apuntador a un objeto de clase enfermoInteligente. Controla los

intervalo y probabilidad de spawn de items, enfermos y del enfermo inteligente, además de sus posiciones de aparición y su eliminación al salir de pantalla.
Controla el scroll del fondo y define una velocidad base con la que se mueve.
Detecta y gestiona las colisiones entre el jugador y los items (obtener efectos) y entre el jugador y los enfermos (perder vida y activar inmunidad).
Adicionalmente escala la dificultad de manera automática aumentando la velocidad del fondo, de los enemigos, las frecuencias de spawn y probabilidades de aparición.
Establece los parámetros de victoria (se recogen 7 ítems) y de derrota (el jugador pierde todas sus vidas)

jugador1

Representa al personaje controlado por el jugador en primer nivel. Se mueve dentro de rangos verticales con una velocidad fija y es arrastrado cuando se queda quieto, su lógica de movimiento se encuentra en el método update, ya que alli se actualiza su posición que es un atributo de tipo QPoint.
Carga los sprites del personaje que tienen un QTimer que los actualiza con los métodos cambiarFrame y draw.
Tiene dos tipos de inmunidad: inmune normal al colisionar con un enfermo estandar e inmune inteligente al recoger un item de tipo 3 que inhibe los ataques del enfermo inteligente.
No contiene los items recogidos, unicamente contiene atributos que llevan un conteo de los items con los que ha interactuado. Así mismo, tiene un registro de 5 vidas que pueden disminuirse.
Devuelve un QRect para detectar colisiones con ítems y enfermos.

enfermo

Representa a los enemigos del primer nivel. Siempre se mueven de manera lineal con una velocidad configurable con el método de aumento de dificultad de nivelPesteNegra.
Carga 3 tipos diferentes de skin que se eligen de manera aleatoria que no cambian el comportamiento de los enfermos y manejan la misma lógica de cambios de frame para dibujar que jugador1.
Expone su área con getRect para colisionar con el jugador. Cuando se verifica su colisión con el jugador en nivelPesteNegra se llama a un método de reducción de vidas de jugador1.

enfermoInteligente

Hereda de enfermo. Representa al elemento inteligente dentro del juego.
El enfermo inteligente aparece recurrentemente en pantalla durante la partida. En cada aparición, mientras se mueve, evalúa durante cierto tiempo al jugador y los ítems en el suelo. Si el jugador tiene más de 3 ítems, está cerca al inteligente y no tiene activa la inmunidad contra el inteligente, decide perseguirlo y robarle ítems. Si no se da alguna de las condiciones, el inteligente prefiere robar el ítem más cercano. El inteligente guarda información sobre su aparición inmediatamente anterior; si intento robar al jugador y triunfo, aumenta su velocidad a la siguiente aparición, en cambio sí intento robar y fallo, reduce el boost de velocidad que adquirido anteriormente.

Tiene varios estados que cambian durante la partida: Caminando, comportamiento normal durante el cual evalúa el entorno y toma decisiones; persiguiendoJugador, cuando se activa sigue la trayectoria del jugador durante 4 segudos; yendoAItem, cuando decide tomar un item del suelo; Huyendo, después de robar escapa aumentado su velocidad; Autónomo, vuelve a caminar cuando termina su periodo de actividad.
El enfermo inteligente analiza constantemente la distancia al jugador, la cantidad de ítems del jugador, los ítems que estén activos en ese momento, los ítems más cercanos al suelo (excepto los de tipo 3) y si el jugador tiene inmunidad especial, mediciones que cambian automáticamente su estado y por ende su comportamiento.
Guarda su posición como QPoin (para calcular los frames y dibujar la animacion) y como QPointF (para moverse en pantalla de manera fluida cuando persigue)
Su movimiento se da de la siguiente manera: Primero calcula la diferencia en x y en y entre él y el jugador y utiliza Pitágoras para calcular la distancia, normaliza el vector dirección y lo escala (multiplica) por la velocidad. Se repite el proceso para ir a un ítem.

item

Representa las hierbas medicinales que el jugador puede recoger.
Tienen dos estados; posado, cuando están sobre el suelo y se mueven horizontalmente hacia la izquierda con la misma velocidad del fondo sin ningún movimiento en Y, y rebotando, cuando spawnean al lado derecho de la pantalla aparecen con cierta altura Y y se calcula "suelo" relativo sobre el cual se dará un rebote debido a la gravedad que acelera la caída y una velocidad horizontal inicial que se reduce con cada rebote por la fricción, además de un coeficiente de restitución que reduce la energía con cada rebote y por ende la altura alcanzada en cada uno de manera progresiva.
Contiene un método que establece aleatoriamente las velocidades en X y Y, el coeficiente de restitución, la fricción y el número de rebotes totales para cada ítem.
Existen 3 tipos de items que se elijen tambien aleatoriamente; los item tipo 1 son aquellos que el jugador recoge y acumula para ganar la partida, los tipo 2 aumentan 1 punto de vida al jugador, y los tipo 3 otorgan inmunidad temporal contra los robos del enemigo inteligente.
Devuelven su zona de colisión con getRect.

nivelColera

Hereda de nivel. Contiene apuntadores a un objeto de clase doctor (jugador) y a objetos de clase proyectil (ampolla y piedra con comportamientos distintos), y listas con apuntadores a objetos de clase obstáculo y balde.
Controla los imputs del mouse para lanzar proyectiles definiendo la trayectoria mediante el arrastre del cursor, el proyectil seleccionado para lanzar y la cantidad de baldes rotos y baldes limpios. Se encarga de calcular la posición de los proyectiles usando un modelo de tiro parabólico considerando gravedad.
También define las condiciones de derrota (se agotó el tiempo o se rompieron demasiados valdes) y victoria (todos los vales de llenaron correctamente)
El movimiento de los proyectiles se calcula de la siguiente manera: Cuando el jugador hace click y arrastra sobre el doctor se captura el punto inicial y actual del mouse y la dirección y fuerza del lanzamiento según cuanto se tire hacia atrás. Luego se calcula la trayectoria con la

fórmula de tiro parabolico del modelo MUA de la física. Adicionalmente los proyectiles tipo ampolla pueden rebotar con un factor de rebote establecido.

doctor

Personaje que puede controlar el jugador en el segundo nivel. No posee movimiento, ya que el jugador únicamente controla la potencia y trayectoria del tiro, no cambia de posición al doctor.
Maneja la lógica completa de los sprites correspondientes a la animación de lanzamiento del doctor segun la etapa del lanzamiento en la que se encuentre, con un QTimer independiente de animación.

proyectil

Hereda de QGraphicsPixmapItem. Representa los objetos lanzables por el jugador. Contiene atributos que controlan registran la visibilidad, posición, velocidad, gravedad y factor de rebote de los proyectiles.
Según el tipo de proyectil (piedra y ampolla) contiene comportamientos específicos representados en los atributos puedeDestruirObstaculos, puedeRebotar y puedeLlenarBaldes.
Contiene referencias a los objetivos con los que puede interactuar; de las clases obstáculo y balde, además de un puntero a nivelColera para calcular las físicas del tiro parabólico.
Se encarga de iniciar el movimiento del proyectil con la velocidad y posición dadas, actualizar el movimiento, verificar las colisiones, aplicar efectos (rebote, destruccion, llenar balde) y manejar los rebotes luego de las colisiones.

objetoEstatico

Proporciona una base para todos los objetos estáticos del nivel que ocupan un espacio físico y pueden interactuar con los proyectiles. Hereda de QGraphicsPixmapItem. Incluye métodos para acceder y modificar su área de colisión.

balde

Hereda de objetoEstatico. Representa los objetivos a los que el jugador debe apuntar.
Puede cambiar de estado entre sucio, lleno (limpio) y destruido. Contiene los métodos que definen las interacciones con los proyectiles y cambian su estado, como llenar y destruir.
Proporciona su área de colisión activa cuando esta sucio.

obstaculo

Hereda de objetoEstatico. Representa obstáculos que forman parte del escenario y deben ser destruidos para poder acceder a los valdes. Sus atributos indican si el obstáculo ha sido destruido y definen su área de colisión. Además, contiene métodos para cambiar su aspecto y estado ante las demás clases una vez que se destruye.

nivelCovid

Hereda de nivel. Contiene un objeto de clase dron (jugador), un objeto de clase baseCarga y vectores con apuntadores a objetos de tipo pajaro y zonasInfeccion.
Se encarga de controlar el spawn de pájaros y zonas de infección por medio de tiempo y probabilidad de aparición para cada uno y los elimina de pantalla y memoria cuando dejan de existir en la lógica del nivel.
Verifica las colisiones entre el dron con los pájaros (disminuyen su batería) y el dron con las zonas de infección (el dron reduce su tamaño y las elimina).
También define las condiciones de derrota (batería del dron agotada, alguna zona de infección supera el radio máximo o se excedió el tiempo límite antes de desinfectar todas las zonas necesarias)

dron

Hereda de QObject. Representa el personaje controlado por el jugador. Puede moverse a través de toda la pantalla, siendo los límites de movimiento los bordes de la misma.
Establece la posición, velocidad, aceleración, velocidad máxima y fricción para el movimiento del dron.
Guarda el estado de batería del dron y los valores de consumo por estar activo y moverse, además del efecto temporal de inmunidad a las colisiones.
Contiene métodos para gestionar la batería, como vargarBateria y quitarBateria y devuelve su hitbox con getRect.
El movimiento del dron se da de la siguiente manera: Se le suma el valor de la aceleración al vector velocidad, se usa Pitágoras para calcular la velocidad resultante como la norma del vector y se normaliza el vector para luego escalarlo dentro del límite de velocidad máxima. Posteriormente se aplica fricción para dificultar el control del dron cuando se necesita precisión en el manejo.

pajaro

Hereda de QObject. Representa un enemigo móvil en el nivel.
Sus atributos establecen su posición, velocidad en X y Y, posición inicial de referencia para movimientos curvos o parabólicos, tipo de movimiento que realizara, tiempo trascurrido desde el spawn y parámetros para movimientos especiales como amplitud, frecuencia o fase.
Contiene metodos que configuran de manera aleatoria dentro de rangos definidos los parametros de movimiento de cada pajaro, como el tipo de movimiento con el que spawneara.
Los pajaros pueden realizar 5 tipos de movimientos: Lineal, parabolico, senoidal, curvo e inestable. Primero se utiliza un metodo para iniciar el movimiento desde los bordes de la pantalla, el cual elije aleatoriamente un borde de aparicion, coloca la pocision del pajaro por fuera de la pantalla y le asigna una velocidad inicial en X y Y hacia el interior de la ventana, le asegura una velocidad minina y ajusta ligeramente su direccion hacia el centro de la pantalla para que no se mueva unicamente en los bordes. Posterior a eso se mueve actualizando su pocision segun el tipo de movimiento: para el movimiento lineal se utiliza la velocidad en X y Y, el movimiento parabolico usa la aceleracion gravitacional en la formula del MUA para el movimiento vertical, el movimiento senoidal consta de un desplazamiento horizontal constante y un desplazamiento vertical con una onda senoidal (se suma la amplitud

y se multiplica el seno de la frecuencia sumada a la fase), en el movimiento curvo se mueve siguiendo un radio creciente alrededor de un punto lo que produce un espiral o giro curvo y finalmente para el movimiento inestable se definen variaciones aleatorias (ruido en X y en Y) lo que produce un movimiento aproximadamente lineal con cambios de direccion y velocidad frecuentes.

zonaInfeccion

Hereda de QObject. Representa áreas de infección que aparecen en la pantalla que pueden crecer y ser desinfectadas por el jugador.
Sus atributos guardan su posición central en la pantalla, el radio actual de la zona, el radio máximo al que puede llegar y su velocidad de crecimiento (expancion del radio).
También se guarda su estado de actividad, si está siendo desinfectada y cuánto tiempo ha pasado en el proceso de desinfección, el tiempo total necesario para desinfectarla y los parámetros visuales de color y pulso.
Contiene métodos que generan su posición de manera aleatoria en la pantalla cuidando que no se superponga con otras zonas y que controlan su estado de infección/desinfección.

baseDeCarga

Representa un elemento fijo en la escena donde el dron puede recargarse cuando gaste su batería.
Sus atributos guardan su posición central en la pantalla y el radio dentro el cual el dron puede empezar a cargarse. Su método principal determina si el dron se encuentra dentro del radio de carga calculando la distancia entre ambos. La clase nivelCovid se verifica si el dron se encuentra en el radio de carga para aplicar el efecto.

Resumen de las mecánicas

Nivel peste negra: El jugador debe esquivar enfermos que tienen un movimiento lineal hacia la izquierda mientras él se mueve hacia la derecha en una pantalla de scroll. Tiene 5 vidas que pierde gradualmente al colisionar con un enfermo y puede recuperar con ítems tipo 2. Debe recoger ítems tipo 1 del suelo para ganar la partida. El nivel contiene al elemento inteligente (enfermo inteligente) que persigue al jugador, roba al jugador, toma ítems del suelo y escapa según la decisión tomada. El jugador puede ganar inmunidad temporal al robo del enfermo inteligente recogiendo ítems tipo 3.
El nivel contiene el modelo físico de rebote de los ítems, y un método de aumento gradual de la dificultad según el tiempo de la partida.

Nivel colera: El jugador debe apuntar y disparar proyectiles a los objetivos. Debe seleccionar las piedras para destruir los obstáculos y las ampollas para llenar los baldes. Debe tener precaución de no desperdiciar la cantidad limitada de ampollas ni romper los baldes con las piedras. El tiempo para desinfectar todos los valdes es limitado.
El nivel tiene el modelo físico del tiro parabólico de los proyectiles y del rebote de las ampollas en los obstáculos, además del temporizador de la partida.

Nivel covid: El jugador debe interactuar con las zonas de infección durante cierta cantidad de tiempo para desinfectarlas al tiempo que debe esquivar a los pájaros que reducen su batería al colisionar con él. La batería del dron se disminuye con las colisiones, el movimiento y el tiempo de actividad, por ende, debe recurrir frecuentemente a su base de carga para no quedarse sin batería y perder el nivel. El nivel cuenta con un temporizador, que marca un límite de tiempo en el cual se debe desinfectar un número fijo de zonas.
El nivel contiene un temporizador y modelos físicos en el movimiento acelerado con fricción del dron y las trayectorias parabólicas, senoidales y curvas de los pájaros.

Guía de instalación y uso

El archivo ejecutable debe descargarse en la computadora y ejecutarse de manera tradicional. Este dirige al usuario al menú principal del juego, cuyos botones funcionan con entrada por mouse.

El primer y tercer nivel se juegan únicamente con entrada por teclado, siendo las teclas A, W, S y D las que deben usarse para controlar el movimiento del personaje horizontal y verticalmente. El segundo nivel se maneja con entrada por mouse y teclado, el jugador debe hacer click en el doctor y arrastrar sin soltar para ajustar la potencia y trayectoria de los proyectiles, además, debe presionar la barra espaciadora antes de ajustar la trayectoria para cambiar el tipo de proyectil que lanzará.

Resultados y discusión

Durante el desarrollo del proyecto se nos dificultaron la implementación de los sistemas físicos principalmente, ya que fueron de los elementos con lógica más compleja que desarrollamos, en especial los límites de movimiento que debía tener cada entidad, lo que agrego otra capa de lógica a las ecuaciones físicas. Tuvimos que desarrollar límites estrictos especialmente en la generación de los parámetros aleatorios para los pájaros del nivel 3, ya que las ecuaciones por si solas generaban movimientos no visibles en pantalla o que excedían los límites, restándole propósito a dicha entidad.
Otro factor que tardo en su implementación fue el enemigo inteligente en el nivel 1, ya que su cantidad de verificaciones y mecánicas diferentes generaron una lógica de movimiento algo tediosa al principio de su desarrollo, lo que necesito gran cantidad de pruebas hasta llegar al producto final.

Podemos concluir que el desarrollo del proyecto fue satisfactorio. Logramos desarrollar a tiempo y de manera correcta todas las mecanices e interacciones que propusimos en el análisis inicial y logramos llegar a un resultado estético, funcional y con buena jugabilidad.

Para un próximo proyecto mejoraríamos la distribución del tiempo que tuvimos y la complejidad de las expectativas de desarrollo que planteamos.

Conclusión

Desarrollar este videojuego fue una forma práctica de recopilar todo lo aprendido durante el curso. Nos permitió aplicar POO, memoria dinámica, físicas y el uso de Qt en un proyecto real, donde cada decisión afectaba directamente cómo se sentía y funcionaba el juego. Tuvimos que pensar en cómo organizar de manera correcta las clases, cómo comunicar los niveles con los menús, cómo manejar colisiones, animaciones y temporizadores, y cómo lograr que todo corriera de forma fluida.

Durante el proceso nos encontramos con varios retos, especialmente al implementar las físicas, el agente inteligente y el manejo de muchos objetos moviéndose al mismo tiempo. Esto nos obligó a revisar el diseño inicial y a mejorar varias partes del código para que el juego fuera estable y entendible.

El resultado final es un juego completo con tres niveles diferentes, cada uno con mecánicas propias, físicas distintas, sonido, menús y un agente inteligente que aporta variedad. Aunque todavía hay cosas que se podrían perfeccionar, como mejorar algunas animaciones o añadir más comportamientos, el proyecto cumplió lo que buscábamos y muestra claramente lo que aprendimos.

En general, el trabajo nos ayudó a entender mejor cómo se construye un software interactivo y lo importante que es planear, probar y ajustar constantemente. También nos dejó más confianza para enfrentar proyectos más grandes en el futuro.