Memoria

A DARK SHADE OF WHITE

Blanca Beceiro Rey
Luis Felipe Llamas Luaces
Jorge Núñez Rivera
Raúl Santoveña Gómez

Índice

1. Desarrollo artístico

1.1. Antecedentes

1.1.1. Ambientación

Nueva York, año 2030, la crisis política, económica y social que vive la ciudad asola las calles. La población se encuentra subyugada bajo las exigencias de un gobierno estéril y en ruinas. Los delincuentes proliferan a sus anchas y parece que ha habido un retorno a la época de los primeros humanos, donde la ley del más fuerte prevalecía. Destacan por su brutal violencia, su arrogancia y su absoluta estupidez.

Sin embargo, los criminales más peligrosos de la ciudad se encuentran sentados en una silla de despacho planeando al mínimo detalle y con el mínimo coste los robos más inverosímiles, las fugas más improbables y provocando las noticias periodísticas más sorprendentes del momento. Son los llamados ladrones de guante blanco.

Estas circunstancias han obligado a lo que queda del gobierno a actuar, ordenando a las autoridades la decisión de reabrir la antigua cárcel de Alcatraz . Una prisión de máxima seguridad adaptada tecnológicamente a los nuevos tiempos donde la fuga es imposible.

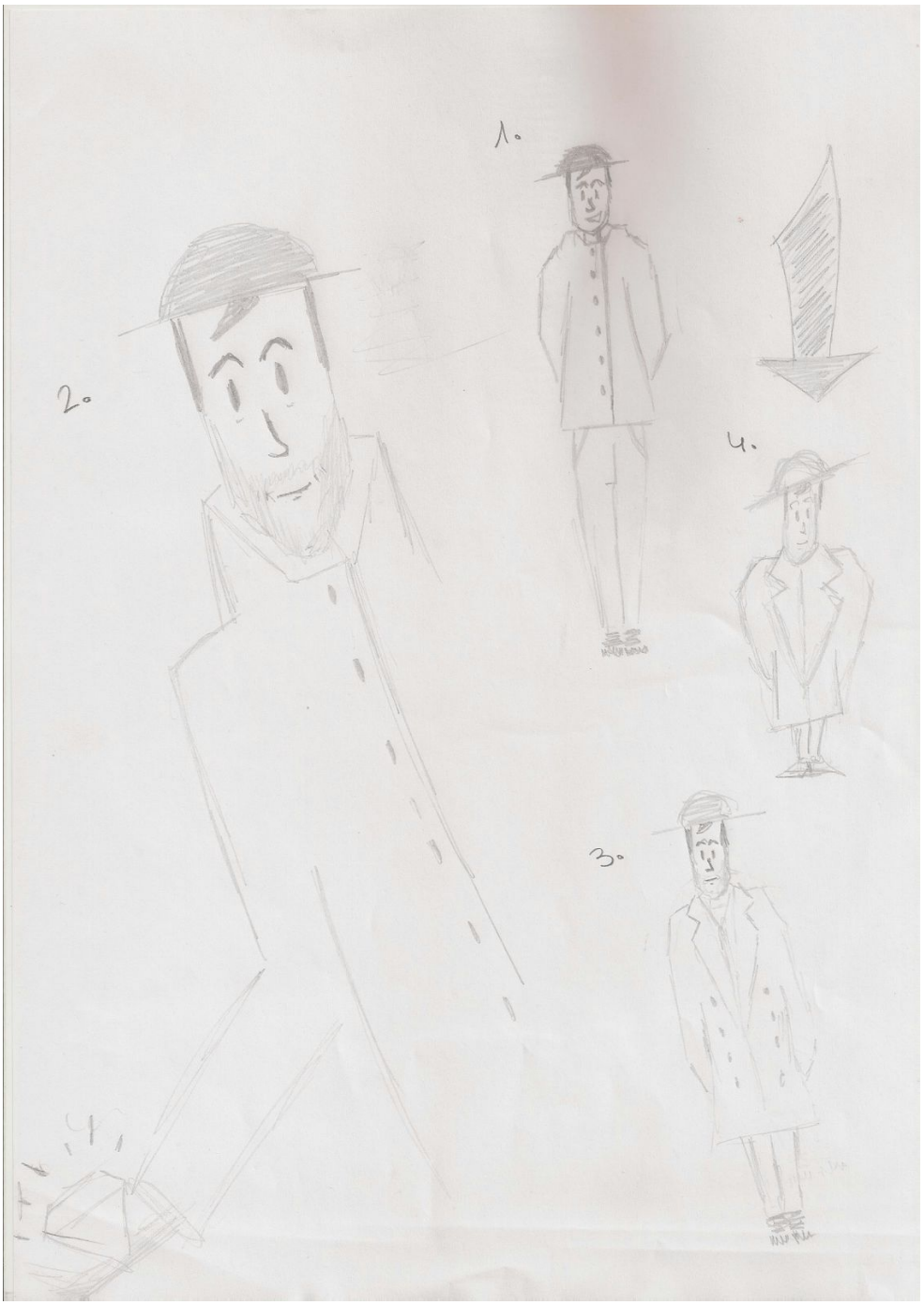
1.1.2. Historia

Ralph L. White es un profesor de informática aparentemente corriente. Pasa los días entre la facultad y su pequeño apartamento en un barrio humilde de la ciudad. Por las noches, su vida es otra. El señor White recorre los centros culturales y económicos en busca de vulnerabilidades aprovechables para su trabajo secreto, los robos.

En uno de sus trabajos, el señor White intenta robar un banco y es detenido y enviado a la prisión de “Nuevo Alcatraz”. Allí, idea un plan para escapar con la ayuda de su antiguo alumno Adrian, su hijo secreto, circunstancia que este último desconoce.

1.2. Personajes

Ralph Lewis White: Protagonista principal. Gran profesor, mejor ladrón. Su trabajo en la universidad no da para pagar las facturas, por lo que decide emprender un camino repleto de robos y delitos.



Bocetos de Ralph L. White

Adrian White: Alumno del señor White en la universidad. No comparten apellido por casualidad. Adrian es el hijo secreto de Ralph, el

cual ha heredado la afición de su padre por la informática. Vive gracias a pequeños timos informáticos. No tiene aparición física en la historia. Sus intervenciones se realizan desde la lejanía, más concretamente desde su ordenador.

John DeBeak: Compañero de crímenes de Ralph, de personalidad cambiante. Decide traicionar a White tras una discusión sobre qué hacer con el cuadro una vez robado.

1.3. Otras características de la ambientación

1.3.1. Objetos destacados

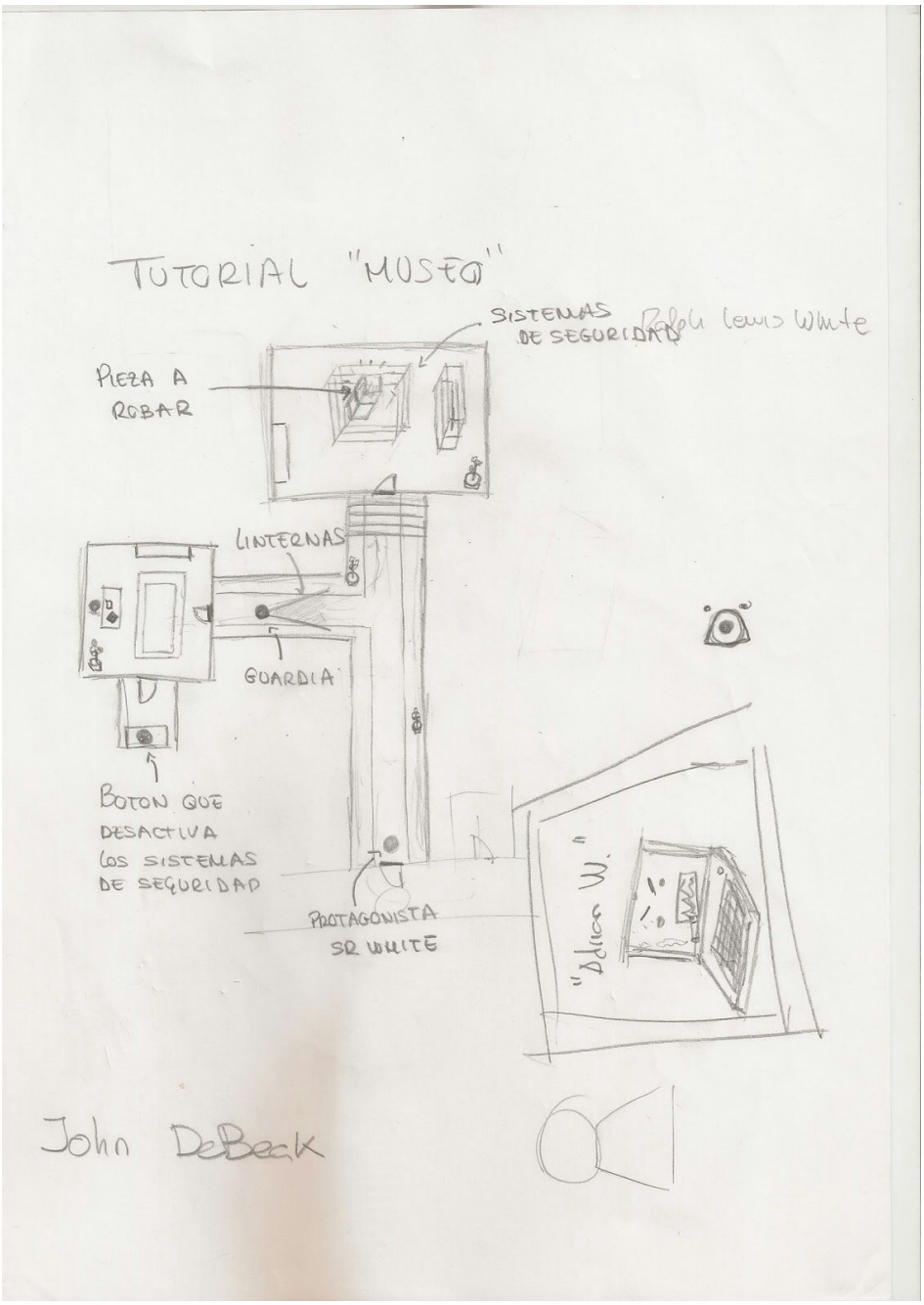
- Black Dot: Pieza de arte valorada en millones de dólares expuesta en el museo de Nueva York. Es literalmente un punto en un folio en blanco.

- La estrella de oriente: Diamante gigante que se encuentra protegido en la cámara más profunda del banco central de la ciudad.

1.3.2. Lugares

- Museo: Un museo de arte, el arte, al contrario que sus sistemas de seguridad es moderno. La gente no sabe si las sillas son para sentarse o si son parte de la exposición. En una de sus salas se encuentra actualmente en exposición el “Black Dot” un cuadro valorado en 5 millones de dólares, a pesar de ser un folio blanco con un punto negro en medio. Parece el blanco perfecto para nuestro protagonista, ya que los sistemas de seguridad distan de ser perfectos, sin embargo el museo cuenta con un pequeño circuito cerrado de cámaras y una cuadrilla de vigilantes.

- Banco: El banco central de Nueva York es una sucursal de máxima seguridad, que alberga grandes cantidades de dinero, joyas, y objetos personales valiosos de los peces gordos de la ciudad.



Bocetos del museo y Adrian White

- Cárcel: Reconstruida a partir de la original, “Nuevo Alcatraz” es la cárcel que ningún maleante quiere visitar. Situada en una pequeña isla cercana a la ciudad, se encuentra totalmente aislada, la seguridad goza de grandes recursos tecnológicos y los guardias están preparados y entrenados para golpear y después preguntar.

Concepto de Nuevo Alcatraz

1.4. Guión

- 2D
  - Primera fase (El Museo)
    - Ralph L. White descubre un fallo en la seguridad del museo de la ciudad, donde se encuentra expuesto el “Black Dot”, por lo que decide robarlo.
    - Ya en el museo, se encuentra con varios pasillos y salas, vigiladas por un guardia de seguridad que recorre el museo. Además, el Black Dot está custodiado por un sensor de movimiento que sólo puede desactivarse desde el cuarto del vigilante de seguridad.
  - Segunda fase (El Banco)
    - Ante el éxito de la misión, el señor White decide asaltar el banco central de Nueva York, donde ha oído que se encuentra el diamante más grande y valioso del mundo, aparte de grandes cantidades de dinero con las que podría vivir cómodamente el resto de su vida.
    - El banco cuenta con una gran sala central, vigilada por varios miembros de seguridad, cámaras que se activan cuando detectan el movimiento y una gran puerta acorazada que protege la habitación donde se encuentra el preciado diamante.

        - Durante la misión, cuando el señor White ya está en poder del diamante, es emboscado por los guardias que fueron avisados por deBeak y detenido. Ante los graves delitos cometidos y sobre todo por el ridículo público al que ha sometido a la seguridad y gobernantes de la ciudad, es condenado a cadena perpetua en la prisión de máxima seguridad de Nuevo Alcatraz.

    - Tercera fase (La Cárcel)

        - Con el señor White entre rejas, Adrian se siente frustrado, era su profesor predilecto en la Universidad, y ahora ya no podrá seguir aprendiendo de aquel al que tanto admira. Enfadado ante la situación, Adrian pretende ayudar al señor White a salir de Nuevo Alcatraz. Una noche, mientras el señor White descansa en su celda, la puerta se abre, las alarmas no han saltado, por lo que Ralph decide aprovechar para intentar escapar.

        - Las seguridad en la cárcel es muy alta, hay guardias en cada esquina equipados con linternas y también cámaras. El diseño no es menos, celdas y más celdas, pasillos pequeños y sinuosos, numerosas habitaciones... y una gran muralla custodiada por francotiradores que hace casi imposible escapar. La única opción es aprovechar el alcantarillado que comunica la cárcel con la ciudad. Pero acceder a él es muy complicado. A sabiendas de que las cloacas son el talón de aquiles de Nuevo Alcatraz, su acceso está fuertemente vigilado y para llegar a ellas es necesario burlar numerosos guardias.

        - Para esta fase, se cuenta con la ayuda de Adrian, que será capaz de abrir puertas, desactivar cámaras, sistemas de seguridad...

- 3D

    Minijuego de control de un robot de infiltración por conductos de ventilación para desactivar los sistemas de seguridad del banco.

2. Desarrollo técnico

2.1 Videojuego en 2D

2.1.1. Descripción

Videojuego de sigilo e infiltración desde una perspectiva ortogonal. El jugador deberá enfrentarse a distintas fases con el objetivo de robar el objeto indicado sin ser atrapado. Para ello tendrá que ser capaz de esquivar las cámaras de seguridad, guardias y demás dificultades presentes.

2.1.1.1. Personajes

- Sr. White: Protagonista de la historia. Es el personaje controlado por el jugador.

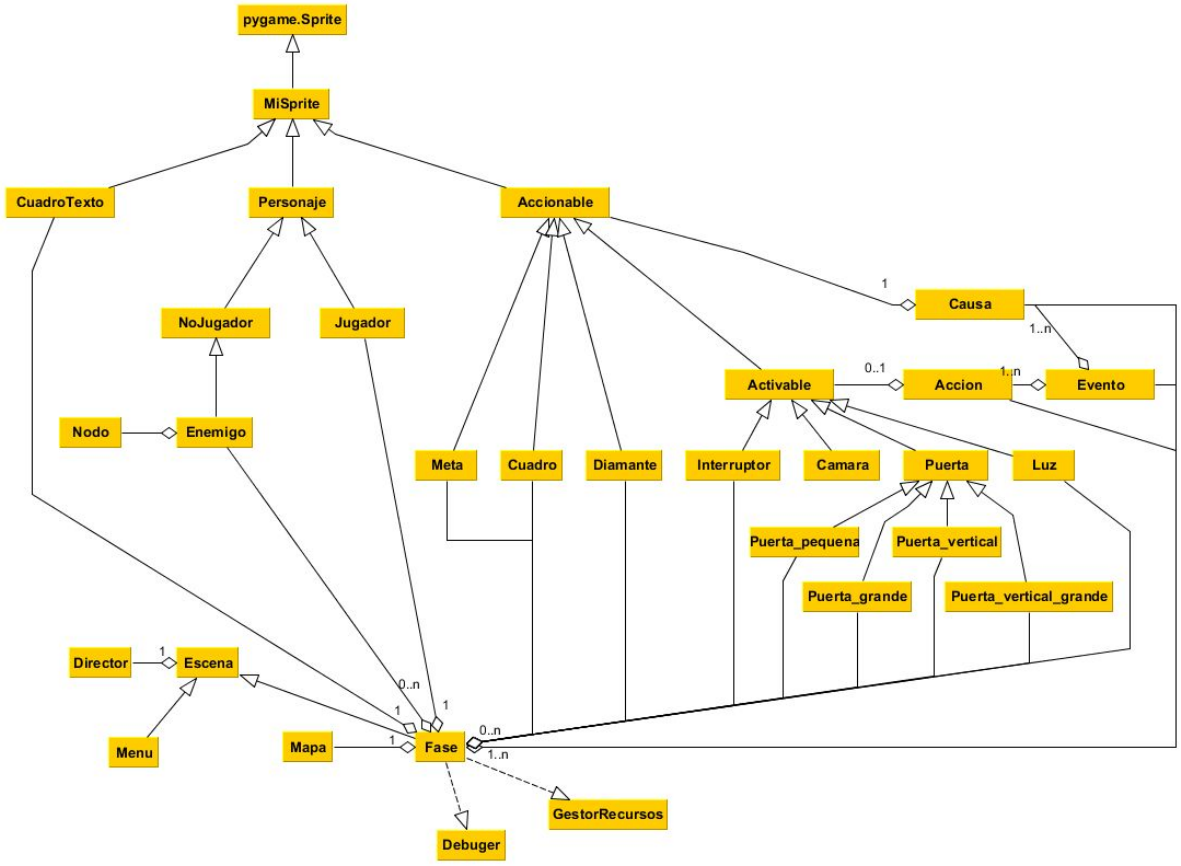
2.1.1.2. Enemigos

- Guardias: Vigilan y custodian los lugares y los objetos valiosos. Patrullan las zonas para detectar intrusos.
- Cámaras: Al igual que los guardias, vigilan determinadas zonas, pero en una posición fija del mapa.

2.1.1.3. Objetos

- Black Dot: Objeto del museo que el jugador tiene que robar.
- La estrella de oriente: Objeto del banco que el jugador tiene que robar.

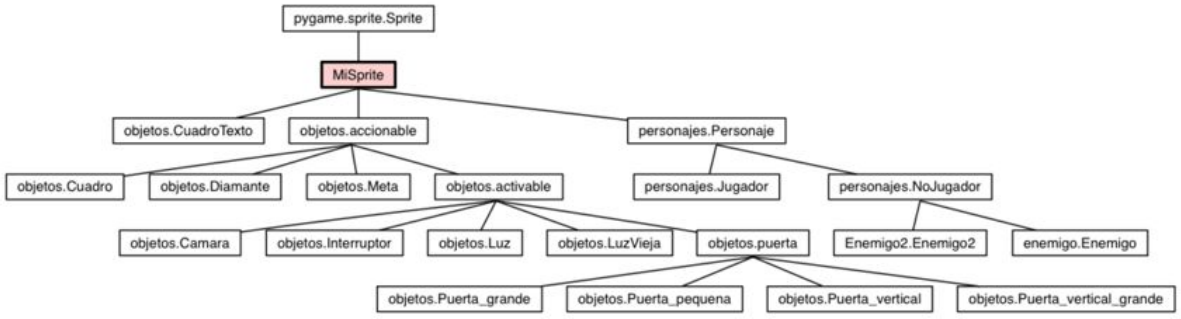
2.1.1.4. Diseño



Clases

MiSprite:

- Deriva de pygame.sprite.
- Contiene métodos para establecer posición, actualizar el scroll y update.
- Update siempre será el método que llamamos en el update y que reciba el tiempo transcurrido para hacer cálculos dinámicos (movimiento, esperas…), en este caso incrementa la posición según la velocidad y el tiempo transcurrido.



CuadroTexto:

- Deriva de MiSprite.
- Contiene métodos para establecer texto y dibujar (draw).
- Es un cuadro estático en el que se renderiza el texto deseado.

Personaje:

- Deriva de MiSprite.
- Representa todos los personajes del juego.
- Contiene métodos para: mover, actualizar postura, update.
- Actualizar postura controla el cambio de frame de la animación.
- Update controla el movimiento y comprueba las colisiones. mover cambia la dirección y velocidad en que se está moviendo.

Jugador

- Deriva de Personaje.
- Representa al personaje controlado por el jugador.
- Sobreescribe el método mover para que se controle con el teclado.

No_Jugador

- Deriva de Personaje.
- Representa una interfaz para los personajes controlados por la cpu.
- Define la función mover_cpu que es la que llamará la fase para actualizar su movimiento.

Enemigo

- Deriva de No_Jugador.
- Contiene: *Estado* (int), *recorrido* (lista de waypoints), *ruta*(lista de waypoints), *rutalocal* (lista de nodos), destino (waypoint), *destinolocal* (nodo), siguiente(waypoint), *siguientelocal* (nodo), *tiempobusqueda*, *tiempopersecucion*, *tiempocalculoruta* (floats), nodos (pares de enteros que representan las posiciones de los waypoints), *grado* (lista de listas de waypoints adyacentes formando un grafo de waypoints).
- Sobreescribe: *mover_cpu* para que vaya siguiendo los waypoints y actualizando su estado.
- Implementa: *estaViendo* (fase,pos,rango) que para una posición objetivo comprueba si es visible para el personaje suponiendo un ángulo de visión (rango), comprobando si en ángulo del vector que apunta al objetivo está dentro del rango y comprobando si ese vector colisiona con el mapa de opacidad alarma (fase, nodo) determina el comportamiento del enemigo cuando suena la alarma recibiendo un waypoint (que es el más cercano a donde se activó la alarma), para que actúe según su estado.
- Sobreescribe update para comprobar los cambios de estado producidos por paso de tiempo (*tiempobusqueda*, *tiempopersecucion*…).

Accionable

- Deriva de MiSprite.
- Contiene *area* (pygame.Rect).
- Representa a todos los objetos que pueden desencadenar eventos, tanto porque el personaje simplemente se encuentre en su área de activación como porque se encuentre en el area y ademas pulse acción.
- Implementa la función *objetoEnArea(Rect_objeto)* que comprueba si un rectángulo se encuentra dentro del area de activación.

Meta, Cuadro y Diamante (Derivadas de Accionable): son clases auxiliares para simplificar la definición de objetos. Simplemente inicializan accionable con una imagen en particular, no implementan ninguna función ni variable.

Activable:

- Deriva de accionable.
- Contiene *estado, tiempocambio.*
- Representa a todos los objetos que pueden cambiar de estado entre dos estados (activado/desactivado), ademas si tienen una animación esta animará el cambio de estado.

Interruptor es una clase auxiliar derivada de activable que la iniciativa con la imagen de un interruptor.

Luz:

- Deriva de activable.
- Sobreescribe *init* y *update* para que en lugar de mostrar una animación. muestre un rectángulo negro y le vaya cambiando la transparencia.

Cámara:

- Deriva de activable.
- Contiene: dirección, *direcciongiro, rangogiro, rangovision.*
- Sobreescribe *update* para que la cámara gire su dirección a medida que cambia su animación, y se pare si su estado cambia a desactivada.
- Sobreescribe *cambiarEstado* para que el cambio sea instantáneo implementa *estaViendo* similar a enemigo.

Puerta:

- Deriva de activable.
- Hace de interfaz para las demás puertas pese a no ser abstracta.
- Sobreescribe *cambiarEstado* para que reproduzca el sonido de la puerta.

Puerta_pequena, Puertavertical, Puerta_vertical_grande y Puerta_grande son clases auxiliares derivadas de puerta que facilitan la creación de objetos inicializando la puerta con una imagen concreta y valores iniciales.

Evento:

- Representa un suceso del juego. Cada evento es un conjunto de 1 a n Causas y otro de 1 a n acciones.
- Implementa comprobar, que comprueba todas las causas y devuelve true si se cumplen todas, y lanzar que ejecuta todas las acciones. La idea es que fase comprueba cada evento y si se cumple lo lanza.
- Contiene:*listaCausas, listaAcciones.*

Causa:

- Representa el desencadenante de un suceso.
- Cada causa está asociada a un objeto accionable.
- Contiene: tipo, objeto.
- *Tipo* es el tipo de desencadenante. puede ser: *Area* (el personaje está en el area objeto)*, acción* (El personaje esta en el area del objeto y pulsa acción), *visto_camara* (el personaje ha sido visto por la cámara objeto).
- Implementa: *Comprobar* (personaje,fase,action).
- Según el tipo desencadenante, comprueba que el objeto asociado esté cumpliendo las condiciones para desencadenarlo.

Acción:

- Representa la acción que se desencadena por un suceso.
- La acción puede estar asociada a un objeto activable.
- Contiene:tipo, objeto, mensaje, sonido.
- El tipo puede ser:
    - Cambiar (cambia el estado del objeto asociado).
    - Mensaje (Muestra un mensaje de texto con mensaje).
    - Sonido (reproduce el sonido sonido).
    - Alarma (Activa la alarma).
    - Fin (Terminas la fase y carga la siguiente).
    - Pillado (Te han pillado y vuelves a empezar la fase).
- Implementa lanzar (fase) que según el tipo de acción realizará sus efectos.

Mapa:

- Representa el escenario del juego. Se carga desde un archivo layers, que contiene información sobre las capas. Contiene las capas que se dibujan antes y después del personaje y las capas invisibles con los mapas de colisiones física y visual.
- Implementa:
    - *Load_layers*: Carga todas las imágenes de las capas a través del gestor de recursos.
    - *Update (scroll)*: actualiza el scroll.
    - paint_all (pantalla) Dibuja todas las capas visibles.
    - *dibujar_pre (pantalla)*:Dibuja las capas marcadas para dibujar antes del personaje.

    - *dibujar_post (pantalla)*: Dibuja las capas marcadas para dibujar después del personaje.
    - *colisionPunto (punto, layer)*: Comprueba en la capa determinada por layer si el color del punto contiene rojo (colisión).
    - *colision (rect, layer)*: Comprueba en una capa si algún de las esquinas de rect tiene rojo en su color (rojo significa colisión).

Nodo:

- Representa los nodos de la búsqueda local del enemigo.
- Contiene su opción, su padre y la distancia del camino hasta él.
- Contiene el método actualizar para cambiar el padre y la distancia de un nodo existente.
- Contiene un método de clase *mejor_nodo (frontera,dest)* que calcula la heurística A* para elegir el mejor nodo en la búsqueda.

GestorRecursos:

- Es una clase estática.
- Carga archivos desde el disco, los devuelve a quien los solicitó y los almacena en memoria por si se le vuelven a pedir no cargarlos otra vez.
- Implementa una función getPath que comprueba que sean correctos los paths este el programa compilado o no.
- Implementa:
    - CargarImagen, CargarArchivoCoordnadas, CargarArchivoSonido, cargarMusica, CargarArchivoFaseJSON, cargarArchivoCapas.
- Todos comprueban si el archivo esta en memoria y lo devuelven, o si no cargan el archivo, lo almacenan y lo devuelven.

Director:

- Es el encargado de manejar el bucle del juego y los cambios de escena.
- Contiene: pila (pila de escenas).
- Implementa:
    - Bucle(escena): Es el bucle de juego, por cada repetición del bucle llama a escena 3 veces para: comprobar eventos, actualizar según el tiempo transcurrido y dibujar, luego hace flip al display.
    - Ejecutar: También es un bucle, comprueba si quedan escenas en la pila y si es así inicia el bucle de la última.
    - *salirEscena*: Sale de la escena y la quita de la pila.
    - *salirPrograma*: vacía la pila y sale de la escena.
    - *cambiarEscena* (escena): Sale de la escena, la quita de la pila y mete la nueva.
    - *apilarEscena* (escena):sale de la escena y coloca la escena nueva por encima (pasa a la escena nueva pero deja la antigua debajo).

Escena:

- Es la interfaz de las escenas que controlará el director.

- Define: update (actualizar según el tiempo transcurrido), eventos (comprobación de eventos y teclas), dibujar (dibujar).

Menu: Es el menu del juego. Contiene el botón de empezar y el de salir.

Fase:

- Es la escena principal del juego.
- Contiene:
    - Diccionarios con:
        - Objetos: todos los objetos (puertas, cámaras, luces, interruptores, metas, cuadro, diamante).
        - Causas, consecuencias y eventos.
        - Lista de waypoints (pares de enteros con las coordenadas).
        - Grafo de los waypoints (lista de waypoints adyacentes a cada uno).
        - Jugador.
        - Grupos de sprites agrupados según sus cualidades:
            - GrupoSprites contiene todos los sprites que se deben dibujar.
            - GrupoSpritesDinamcos contienen todos los sprites que se deben actualizar con el tiempo (update).
            - GrupoColisionables contiene todos los sprites con los que se puede comisionar físicamente (puertas).
            - GrupoJugadores contiene al jugador.
            - GrupoEnemigos coniene a los enemigos.
            - GrupoOpacos contiene a los sprites que comisionan con las lineas de detección visual (puertas, luces).
            - decorado (Mapa).
- Este conjunto de objetos, eventos, enemigos, waypoints y mapa constituye cada fase y se carga todo desde un archivo .json que es un diccionario cuyas entradas corresponden a la información para cada tipo de elementos; si no hubiera elementos habría una lista vacía.

Métodos y funciones de la fase:

- **ActualizarScroll(jugador):** Comprueba la posición del personaje y la posición del scroll, si el personaje se sale del borde y puede mover el scroll lo mueve y si el scroll llega al borde de la fase se queda ahí.

- **update(tiempo):** Comprueba si esta pausado y sino llama a *mover_cpu* de los enemigos, a *update* para todos los sprites dinámicos, enemigos y jugador, comprueba las colisiones con el enemigo y llama a Actualizar Scroll.

- **dibujar(pantalla):** Dibuja la parte previa del decorado, luego los sprites y por último la parte posterior, y en el caso de necesitar mostrar un mensaje dibuja también el cuadro de texto.

- **dispararAlarma():** Llama a *alarma(waypoint)* para todos los enemigos con el waypoint más cercano al jugador.

- **colision(rect):** Comprueba que las cuatro esquinas de rect no colisionen, para ello comprueba colisión con la capa de colisiones físicas del mapa y también con todos los sprites del grupo *SpritesColisionables.*

- **listaRectangulosColisonables()** es una función auxiliar para colisión que devuelve la lista de los rectángulos de los sprites colisionables.

- **colisionLinea(origen,destino,step,capa):** Comprueba si se produce colisión en una línea entre los puntos origen y destino , para hacerlo calcula puntos intermedios de la línea y lo comprueba en la lista de sprites opacos y en la capa pasada como parámetro.Esta función es muy importante porque determina cuando los enemigos y las camaras ven al jugador, es usada por la función *estaViendo.*

- **nodo_mas_cercano(pos,nodos):** Devuelve el waypoint más cercano a una posición cualquiera .

- **calcular_ruta_Anchura(origen,destino):** Calcula una ruta (lista de números de waypoints) entre el waypoint origen y el waypoint destino. Es un algoritmo de búsqueda en anchura.Comprueba si existe ruta entre el waypoint origen y el destino y devuelve la lista de waypoints, si no es posible trazar una ruta, la lista será vacía. Tiene en cuenta las puertas cerradas. Lo usa el enemigo para acudir a la alarma y para volver a su patrulla.

- **calcular_ruta_local(origen,dest):** Calcula una ruta local entre dos puntos origen y destino (coordenadas). Utiliza la clase *nodo* y la función *calcular_nodos_adyacentes* para generar nodos que representan puntos en el mapa. Es un algoritmo de búsqueda A*. El enemigo usa esta búsqueda para perseguir al jugador a lo largo del mapa.
- **buscar_nodo_visitado(nodo,visitados):** Busca un nodo en la lista de los visitados y en caso de encontrarlo devuelve el nodo antiguo para actualizarlo (función auxiliar de *calcular_ruta_local*).

- **calcular_nodos_adyacentes(modo,step):** Función auxiliar de *calcular_ruta_local* que calcula los nodos adyacentes a un nodo. para eso utiliza la función *colisionLinea* para comprobar que no haya obstáculos en la ruta del rectángulo de colisión del enemigo entre el nodo origen y el nodo nuevo, y si colisiona no genera el nodo, intenta generar 4 nodos circundantes. La distancia entre nodos la define la constante SEARCH_STEP

mientras que el step de las líneas que comprueban la colisión la define RAY_STEP. un SEARCH_STEP demasiado grande no pasa por puertas o lugares estrechos y uno demasiado pequeño genera demasiados nodos, mientras que un RAY_STEP demasiado grande atravesaría las paredes (el grosor mínimo de las paredes es 14 pixeles).

- **buscar_nodo_mas_cercano(origen,nodos):** Esta función es similar a *calcular_ruta_local* solo que comprueba la llegada a su destino con la función *comprobar_llegada_nodos* que comprueba si una posición a la que se ha llegado esta muy cerca de cada uno de los waypoints y en caso de coincidir con uno lo devuelve. Esto permite buscar la ruta sin saber el destino, el primer waypoint que encuentre será el elegido. Devuelve el número del waypoint y la ruta.

- **comprobar_llegada_nodos(pos,nodos):** Funcion auxiliar de *buscar_nodo_mas_cercano* que compara una posición alcanzada con todos los waypoints y si está muy cerca de alguno devuelve el número del waypoint.

- **eventos(lista_eventos):** Comprueba los eventos:
  - Comprueba las teclas.
  - Mueve al jugador.
  - Comprueba todos los eventos de la fase.
  -
- **mostrarMensaje(texto):** Muestra un cuadro de texto y pausa hasta que pulses intro.

- **hay_persecucion():** Comprueba si alguno de los enemigos esta persiguiendo al jugador, y en ese caso las acciones se ven penalizadas, habiendo un 25% de probabilidades de ser capaz de llevarlas a cabo.

La IA del enemigo:

La IA del enemigo es bastante compleja ya que combina dos sistemas distintos de cálculo de rutas y tiene muchos estados distintos y transiciones.

Como utilizando waypoints el enemigo no podía perseguir al jugador por la fase por culpa de que chocaba , y utilizando nodos el juego se quedaba parado 3 segundos calculando rutas, se implementó un sistema híbrido que utiliza los waypoints para las patrullas, acudir a las alarmas y volver a las patrullas, y utiliza los nodos calculados en el momento para perseguir al jugador y para volver a los waypoints.

El enemigo tiene 6 estados:

1. Patrullando: Cada enemigo se iniciativa con un recorrido que es una lista de waypoints.Mientras esté en estado patrullando recorre esa

lista de waypoints y cuando la termina vuelve a empezar por el primero.
2. Deambulando: El enemigo camina aleatoriamente por los waypoints y al llegar a uno elige aleatoriamente entre los adyacentes.
3. Yendo a la alarma: El enemigo calcula una ruta desde su próximo waypoint al waypoint de la alarma y sigue esa ruta.
4. Volviendo a la patrulla: El enemigo calcula una ruta desde su waypoint hasta el comienzo de su recorrido y sigue esa ruta.
5. Persiguiendo: El enemigo se sale de los waypoints y calcula una ruta local para perseguirte por el mapa y te persigue comprobando la ruta cada cierto intervalo de tiempo.
6. Volviendo a un nodo: El enemigo calcula el waypoint accesible más cercano y la ruta local a el por el mapa y sigue esa ruta.

El enemigo comienza en estado Patrullando, y permanecerá en este estado hasta que suene la alarma o vea al jugador. Las transiciones serán por tanto:

- En cualquier estado:
    - Si ve al jugador pasa a *5-Persiguiendo* y calcula la ruta local hacia ti (si ya está persiguiendo simplemente recalcula la ruta).
    - Si colisiona durante un tiempo predefinido pasa a *6-volviendo al nodo* y busca el camino por el mapa al waypoint más cercano.
- **Patrullando**
    - Si suena la alarma pasa a *3-yendo a la alarma* y calcula la ruta waypoint más cercano a la alarma.
- **Deambulando**
    - Si pasa cierto tiempo pasa a *4-Volviendo a la patrulla* y calcula la ruta al principio de su recorrido.
        - Si no puede volver sigue deambulando.
    - Si suena la alarma pasa a *3-yendo a la alarma* y calcula la ruta waypoint más cercano a la alarma.
- **Yendo a la alarma**
    - Si suena la alarma en otro sitio recalcula la ruta.
    - Si llega a su destino pasa a 2-Deambulando.
- **Volviendo a patrulla**
    - Si suena la alarma pasa a 3-yendo a la alarma y calcula la ruta waypoint más cercano a la alarma.
- **Persiguiendo**
    - Si te pierde de vista, tras un tiempo,pasa a *6-Volviendo al nodo.*
- **Volviendo al nodo**
- Si suena la alarma y el guarda no la puede atender(porque está en *5-Persiguiendo o 6-Volviendo al nodo*) almacenará el waypoint de la alarma y cuando vuelve a estar en un waypoint acudirá a ella.

- Si el enemigo esta en *3-yendo a la alarma o 5-persiguiendo* irá corriendo y si no se moverá andando.

2.1.1.5. Guión

El jugador empieza la primera fase en un museo, donde tiene como objetivo robar el Black Dot, un cuadro valorado en millones de dólares.

Si la misión se cumple con éxito, el jugador llega a una segunda fase, cuyo objetivo es “La estrella de oriente”, un diamante guardado en las cajas de seguridad de un banco de la ciudad.

2.1.1.6. Reglas

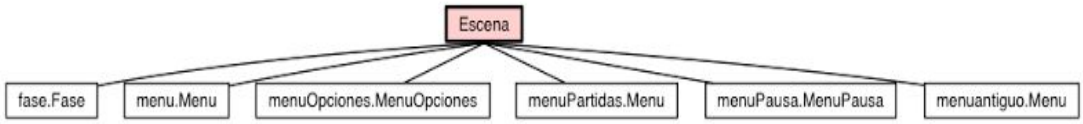
El jugador debe ser capaz de lograr los objetivos sin ser atrapado. Si los guardias son capaces de llegar hasta él el juego finaliza.

El juego se completa si el jugador es capaz de pasar todas las fases con éxito.

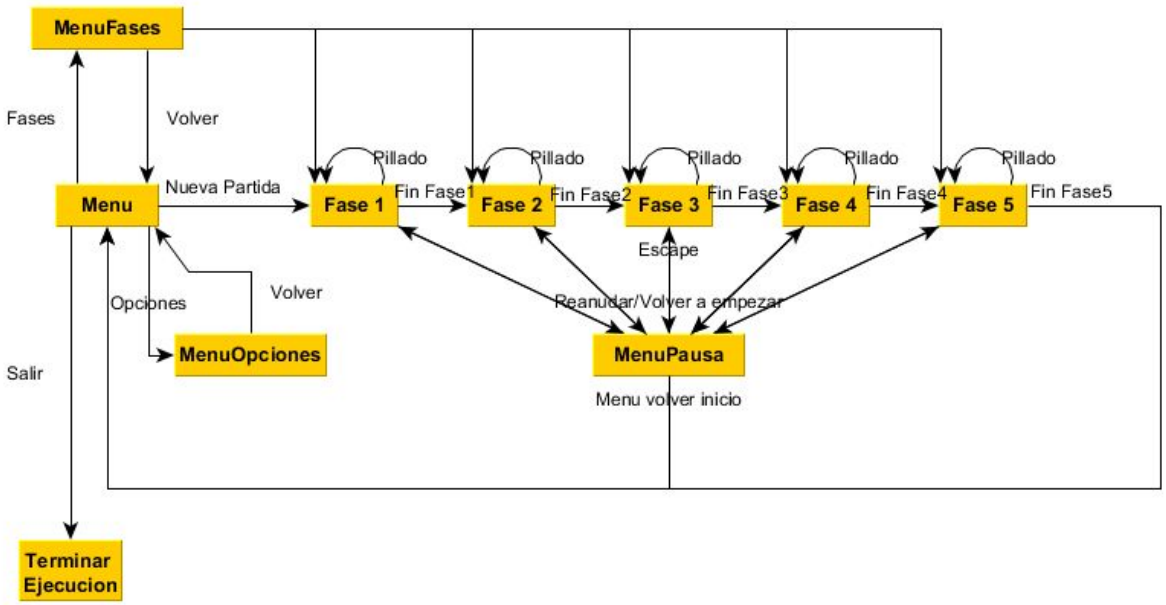
Si una cámara detecta al jugador en la primera fase del museo, los guardias son alertados y van a por él. En cambio, en el banco, el jugador pierde de manera instantánea.

En caso de haber sido detectado por los guardias, la habilidad de interactuar con los diferentes elementos del juego del protagonista se ve minada, habiendo un 75% de probabilidad de fallar la acción.

2.1.2. Escenas



- El juego está organizado en fases, que son clases derivadas de escena. Las mecánicas del juego no varían demasiado entre las diferentes fases, y esas variaciones son implementadas mediante eventos definidos en el archivo JSON de cada fase.
- Los diferentes menús también son derivados de escena, y al contrario que las fases, todas sus funcionalidades son definidas en código y no en archivos externos



2.1.2.0. Escena 0: Menu

2.1.2.1.1. Descripción

Menú principal del juego, permite empezar una nueva partida, desde la primera fase, o bien comenzar ya desde otra diferente.

El submenú opciones está implementado, pero vacío.

En caso de que el jugador presione la tecla escape accederá a el submenu de pausa, que permitirá al jugador reanudar la partida, volver a comenzar la fase o volver al menú principal.

2.1.2.1. Escena 1: Fases del juego

2.1.2.1.1. Descripción

El juego se divide en dos escenarios, el museo y el banco, que cuentan con 2 y 3 fases respectivamente

El museo cuenta con distintas salas conectadas donde se exponen las obras de arte. Hay cámaras colocadas estratégicamente para detectar a cualquier intruso. Además, cuenta con distintos vigilantes de seguridad que patrullan las zonas para asegurarse de que nadie pueda violar la seguridad. También existen puertas sólo activables desde la sala de seguridad, situada en una esquina del museo.

El museo consta de dos fases, la zona de mantenimiento y la sala de exposiciones en si. La zona de mantenimiento se pensó originalmente como un “escaparate” de las diferentes mecánicas del juego, para que el jugador se fuera familiarizando con los conceptos.

Al igual que el museo, el banco cuenta con vigilantes y cámaras de seguridad. Numerosas salas y pasillos esconden la habitación donde se encuentra el diamante que el jugador tiene que robar. El banco está dividido en 3 escenarios, la entrada, una zona de despachos y la cámara de seguridad.

Una peculiaridad del banco es que en el caso de que el jugador sea detectado por una cámara, la partida termina, al contrario que en el banco, que en el caso análogo hace que los guardias procedan a investigar la zona en la que se vió al jugador por última vez

2.1.2.1.2. Diseño de niveles

Todos los niveles se han creado con el editor de mapas Tiled, y posteriormente han sido editados con Adobe Photoshop o Gimp.

Tiled permite la exportación de mapas por capas, lo que ayuda a su dibujado secuencial.

A partir del mapa generado en Tiled, se realizaron dos capas “técnicas” *Colisiones* y *Visible*.

La primera indica por donde se pueden mover los diferentes personajes, y la segunda la opacidad de los objetos con respecto a la visión de la IA. Por ejemplo, una mesa baja permite ver a través de ella, pero sin embargo no se puede caminar por encima.

2.1.3. Aspectos destacables

El motor del juego permite desarrollar fases en ficheros JSON, lo que permite crear nuevos niveles a gente con poco o ningún conocimiento sobre programación.

En el archivo de fase se definen mediante un sistema de coordenadas las posiciones de los diferentes sprites que se pueden utilizar para crear niveles y sus características. También se define la fase siguiente, con lo cual el enlazado entre fases es muy simple.

Se ha generado una documentación en html que se puede encontrar en /doc/html/index.html. Esta documentación se ha generado con la herramienta Epydoc.

2.1.4. Manual de usuario

- Las teclas de control son:
  - Para movimiento: W A S D
  - Para interactuar: Enter

- Para movimiento rápido: Mayúsculas Derecho
- Para salir del juego: Escape

La interacción con el menú se realiza con el ratón

En el directorio doc se ha incluido un video (Demostración ADSOW.mp4) en el que se muestra una posible solución a todas las fases del juego. Por razones de espacio, el video se muestra a 4x su velocidad original.

Para ejecutar el juego, en la carpeta source se ejecutará el comando python main.py

2.2 Videojuego en 3D