Se tiene un Room como **habitación inicial**, desde la cual se puede acceder a un **acantilado** que seria otro Room del cual no se puede salir. Los **fragmentos de objetos de barro, esqueletos**, cofres y demás objetos se pueden modelar al igual que los cofres, cajas, palos, tesoro de los juegos anteriores (extendiendo ConcreteGameObjectHasParent o HasChildren o los dos). Los **frutos venenosos** serían similares al veneno, solo que sin el BooleanState que se prende cuando el jugador sale del cuarto para indicar que el veneno hizo efecto (es decir, solo indicarían que está envenenado). Se puede cambiar fácil si el jugador se envenena al abrir lo que contiene el veneno o al agarrarlo, cambiando en el veneno que acción del jugador es la que lo activa. Los **frutos antídoto** serían idénticos al antídoto, cambian el estado del booleano que indica envenenado. Para las interacciones con estos objetos no es necesario crear ninguna acción mientras que impliquen “pick”, “open”, “look” y “use” que son acciones ya existentes.

El **mono**, si no se planea extender a una inteligencia artificial o entidad que hace algo todos los turnos igual que el jugador, se puede modelar con una simple cantidad de “reglas” medio hardcodeadas que definen su comportamiento (en definitiva una inteligencia son muchas y complejas reglas, si el mono solo hace lo que se dice aca esta parece una solución viable). Las reglas serian literalmente:

- Si mono == despierto y jugador activa mecanismo1 -> activar mecanismo2
- Si mono == despierto y jugador activa mecanismo2 -> activar mecanismo1
- Si jugador no esta con el mono, robar el disco de mas arriba de algún pilar (asumo que en este momento el mono se escapa con el disco y no vuelve, no que sigue volviendo y robando mas)

Donde despertar al mono seria prender un booleano cuando el jugador interactua con el (con “look” o “talk”, que llaman a “be looked” o “be talked to”, acciones que ya están implementadas y funcionan con cualquier par de GameObjects). Estas reglas se chequean luego de cada turno del jugador al igual que las win y loose conditions.

La **puerta con dos mecanismos** seria en realidad tres objetos: la puerta con dos BooleanState que indican si se puede abrir al igual que las LockedDoor, y dos mecanismos con los que se puede interactuar, cada uno con uno de esos BooleanStates. Expandir todo el motor a manejar tiempo, y ver si estoy accionando un mecanismo y apenas hago otra cosa registrar que ya no lo estoy accionando seria molesto y trabajoso. Para lo pedido en la simulación alcanza con que los dos mecanismos se conozcan (y el BooleanState del otro), y cuando activo uno se prenda su booleano y se apague el del otro (simulando que si acciono uno y luego el otro, tuve que alejarme y soltar el primero. como resultado de esta simplificación si acciono uno y luego me voy a abrir un cofre, va a seguir registrando que esta accionado, pero el resultado es el mismo ya que apenas voy a accionar el otro se apagaría el primero). Si acciono uno y encontré al mono, van a estar los dos BooleanStates prendidos y la puerta me va a dejar hacer open. La accion “activar mecanismo” habría que hacerla pero es trivial ya que solo avisa al mecanismo que lo accionaron (similar a “talk”).

La siguiente **habitación con el río y discos** tiene la parte simple de wolf, sheep and cabbage (es decir moverse entre “orillas” que pueden ser cuartos distintos y listo, sin ninguna consideración especial de cuando puedo moverme. esto se puede reusar sin problemas), y las torres de hanoi casi igual (con más discos, pero las reglas son iguales por lo que se puede reusar todo). Lo único que seria necesario es

poder tomar los discos para cruzar, lo cual resulta simple en nuestro modelo porque solo implica agregarle al jugador en su lista de acciones la accion “pick” (la misma que en fetch quest, etc. sin ningún cambio). El limite de llevar un disco por vez se puede poner fácilmente en el inventario del jugador, igual que en treasure hunt.

Para **abrir la segunda puerta** con el disco, se puede hacer que ese disco de mas abajo extienda tanto Disc como Key y que la puerta sea una simple LockedDoor, dependiendo de que tan literal se tome “dejar la llave en el mecanismo de apertura”. Si se debe “introducir” la llave, luego abrir, y luego si quiero agarrarla de nuevo habría que crear un nuevo objeto que permita esto pero seria tan simple como expandir LockedDoor para que tenga una lista de llaves (por si queremos mas de una) y en vez de chequear que el jugador tenga la accion “desbloquear tal código de puerta”, se fija que las llaves estén puestas.

El **arqueólogo** podría ser igual que el ladron solo que en vez de tomar todo toma solo el disco si le hablamos, o incluso se podría separar en dos la accion para que al hablarle informe al jugador que le puede dar el disco con la accion “give”, y el jugador puede elegir hacerlo o no (Give extendería a Move al igual que Pick, Take, Leave o cualquier accion que mueve objetos de un lugar a otro).

Finalmente las win or loose conditions se modelarían con ifs entre los objetos del dominio del juego igual que en treasure hunt o cualquier otro.

Loose:

- if (poisoned.isTrue() or !arqueologo.contains(discoLlave)) and exit.contains(player)

Win:

- if (!poisoned.isTrue() and arqueologo.contains(discoLlave)) and exit.contains(player)