Q1

En este problema, se definen una serie de funciones para trabajar con lógica proposicional, empleando las clases Expr y PropSymbolExpr. Resumiendose:

1) $(A \vee B) \wedge (\neg A \leftrightarrow (\neg B \vee C)) \wedge (\neg A \vee \neg B \vee C)$
2) $(C \leftrightarrow (B \vee D)) \wedge (A \rightarrow (\neg B \wedge \neg D)) \wedge (\neg(B \wedge \neg C) \rightarrow A) \wedge (\neg D \rightarrow C)$
3) Problema temporal con Pacman, usando símbolos temporales:
    a. Pacman está vivo en el instante $t = 1$ si y solo si estaba vivo en el instante $t = 0$ y no due asesinado, o no estaba vivo en $t = 0$ y nació
    b. Pacman no puede estar vivo y nacer al mismo tiempo $t = 0$
    c. Pacman nace en $t = 0$
4) $findModel$ convierte la expresión a CNF y emplea $pycoSAT$ para encontrar un modelo que satisfaga la expresión.
5) $findModelUnderstandingCheck$ ilustra la estructura interna de Expr, devolviendo un diccionario con $\{a: true\}$ para una variable a.
6) $entails(premise, conclusion)$ verifica si una premisa implica una conclusión evaluando si $premise \ \wedge \ \neg conclusion$ es insatisfacible
7) $plTrueInverse(assignments, inverse\_statement)$ retorna $True$ si $\neg inverse\_statement$ es verdadera bajo un conjunto de asignaciones, utilizando pl_true.

Q2

En este problema, definimos una serie de restricciones de cantidad sobre literales lógicas en CNF. $atLeastOne(literals)$ asegura que, al menos un literal es verdadero: $\vee l_i$.

$atMostOne(literals)$ garantiza que, como máximo un literal es verdadero, prohibiendo que pares lo sean de forma simultánea: $\bigwedge(\neg l_i \vee \neg l_j)$.

$exactlyOne(literals)$ combina ambas condiciones para que exactamente un literal sea verdadero: $atLeastOne \wedge atMostOne$

Q3

Este problema modela el movimiento de Pacman usando lógica proposicional y SAT solvers. $pacmanSuccessorAxiomSingle$ y $SLAMSuccessorAxiomSingle$ definen axiomas de sucesión considerando movimientos válidos e ilegales. $pacphysicsAxioms$ impone restricciones físicas: Pacman no puede estar en paredes, debe ocupar una casilla y tomar una acción por timestep, incluyendo axiomas de sucesión y sensores. $checkLocationSatisfiability$ construye la base de conocimiento y consulta al SAT solver para verificar si Pacman puede o no estar en una posición específica en un timestep dado.

Q4

Se implementa un razonamiento paso a paso para planificar el movimiento de Pacman usando lógica proposicional. Primero, se declara la **posición inicial** de Pacman en el tiempo 0. Luego, para cada timestep t hasta 50, se añade a la base de conocimiento (KB) que Pacman debe estar **exactamente en una casilla** y tomar **exactamente una acción**. Se generan los axiomas de sucesión para todas las posiciones posibles, y se consulta al SAT solver si Pacman puede alcanzar la posición objetivo; si existe un modelo que lo permita, se extrae la secuencia de acciones correspondiente. Esto permite construir un plan paso a paso garantizando que se respeten las reglas del juego y las paredes del tablero.

Q5

Se implementa un razonamiento paso a paso para que Pacman coma toda la comida usando lógica proposicional. Primero, se declara la **posición inicial** de Pacman y la ubicación de cada comida en el tiempo 0. Luego, para cada timestep *t*, se añade que Pacman está **exactamente en una casilla**, toma **exactamente una acción**, y se generan los **axiomas de sucesión** para todas las posiciones posibles. Además, se codifica la **dinámica de la comida**: si Pacman está en una casilla con comida, esta desaparece en el siguiente timestep; si no, permanece. Se consulta al SAT solver si toda la comida ha sido comida, y si existe un modelo, se extrae la secuencia de acciones que logra este objetivo.

Q6

Se implementa un algoritmo de **localización de Pacman** usando lógica proposicional y razonamiento paso a paso. Primero, se inicializa la base de conocimiento (KB) con la ubicación de las paredes, marcando correctamente las casillas que son paredes y las que no. Luego, para cada timestep *t*, se añaden a la KB los **axiomas de física**, las acciones tomadas por Pacman y las reglas derivadas de los **perceptos** del agente. A continuación, se consulta al SAT solver para determinar las **posibles posiciones de Pacman** en ese tiempo, y se agregan a la KB las posiciones que se pueden inferir como verdaderas o falsas. Finalmente, el agente actualiza su estado interno y se devuelve la lista de posiciones posibles en cada timestep mediante un generador.