IA02
Solveur Helltaker

Mallarme Corentine
Ouédraogo Taoufiq
Vaurs Damien

Projet réalisé dans le cadre de l'UV IA02, enseignée à l'UTC par Sylvain Lagrue et Khaled Belahcene.

Table des matières

Introduction

> "You woke up one day with a dream. Harem full of demon girls. You've opened the portal in hopes of fulfilling your wildest desires. Hellfire burns through your lungs, death awaits around every corner and everything looks like from a cutesy mobile game. You are in hell." **- Helltaker**

Helltaker est un jeu vidéo composé de dix niveaux. Il s'agit d'un jeu de puzzle de la famille du Sokoban. Le joueur incarne un personnage pouvant se déplacer dans un environnement 2D et dont le but est de rassembler tous les démons des enfers. Pour se faire, il doit résoudre un puzzle et atteindre le démon du niveau. A la manière du Sokoban, il ne peut pas tirer des objets mais seulement en pousser. Le jeu est disponible gratuitement sur la plateforme Steam, pour MacOS, Linux et Windows.

Dans le cadre de l'UV IA02, nous avons réalisé un projet qui a pour but de créer différentes "IA" qui seront capables de résoudre un niveau de Helltaker en renvoyer la séquence d'actions à réaliser pour valider un puzzle du jeu. Ce projet nous permettra de mettre en pratique les notions et les implémentations de la logique étudiées en cours.

Dans un premier temps, nous mettrons au clair les règles du jeu Helltaker et nous modéliserons le problème en STRIPS. Puis nous utiliserons deux méthodes différentes afin d'implémenter une solution au problème : la rechercher dans un espace d'état grâce au langage python, et l'utilisation d'ASP. Après avoir décrit ces méthodes, nous terminerons ce rapport en les comparant sur différents critères de performance afin d'évaluer la méthode la plus pertinente dans le cadre du jeu Helltaker.

1 Préliminaires

La première étape de notre projet a été la modélisation du problème à résoudre. N'ayant pas de manuel sur les règles du jeu, nous les avons donc testées directement sur le logiciel afin de définir les actions possibles et leurs conséquences.

1.1 Fonctionnement du jeu Helltaker

1.1.1 Types d'objets rencontrés

Afin d'expliciter au mieux les règles du jeu, nous utiliserons le vocabulaire suivant pour parler des différentes types de cases et d'objets. Nous avons essayé de trouver des nominations les plus claires possibles.

- **Helltaker**
- **Case but** (démon, ange ou porte du diable), **Case vide** et **Mur**
- **Clef** et **Cadenas**
- **Piège à ours** (peut changer d'état)
- **Objets déplaçables** : Rocher et Squelette (peut être éliminé)

1.1.2 Règles du jeu

Suite à de multiples tests sur le jeu, nous avons pu déduire les règles suivantes et nous utiliserons ces dernières comme base pour la modélisation et l'implémentation de nos solutions. Tout d'abord, le but de Helltaker est d'**atteindre une des cases horizontales ou verticales adjacentes à la case but**. Pour chaque niveau, un compteur indique le nombre maximum d'actions réalisables. S'il atteint -1, la partie est perdue.

Helltaker peut se déplacer de manière verticale ou horizontale sur une case vide (ou une case clef). Chaque déplacement coûte un point au compteur s'il est possible. Il peut aussi se déplacer sur une case contenant un piège à ours. Pousser ou se cogner contre un mur ne coûte pas de point.

Helltaker peut **pousser un rocher ou un squelette** de manière verticale ou horizontale s'il se trouve sur une case adjacente. L'objet se déplace si la case suivante est vide, s'il s'agit d'une case clef ou s'il s'agit d'une case piège à ours. Que l'objet se déplace ou non, l'action coûte un point au compteur. Lorsqu'un objet est poussé vers un mur ou la case démone, alors il ne se déplace pas mais le coût de l'action reste un point du compteur.

Pousser un squelette contre un mur ou contre un rocher **détruit le squelette**. Les rochers ne peuvent pas être détruits. Le coût pour détruire un squelette est le même que pour pousser le squelette.

Les **pièges à ours** peuvent rester ouverts tout le niveau ou s'ouvrir et se fermer à chaque fois qu'une action est réalisée (c'est-à-dire à chaque fois que le compteur baisse). Lorsqu'Helltaker passe ou reste sur une case piège à ours qui est ouvert, alors cette action **coûte deux points** au compteur au lieu d'un.

Helltaker ne peut pas pousser **les clefs** : il doit passer sur la case pour les récupérer. Les cadenas ne s'ouvrent que lorsqu'Helltaker est en possession d'une clef et passe sur leur case. Le cadenas disparait alors. Cette action coûte un point comme s'il s'agissait d'un déplacement simple. Si Helltaker pousse ou tente de se déplacer sur un cadenas sans avoir récupéré la clef, alors il ne se passe rien mais le compteur perd un point. Ouvrir les cadenas n'est pas obligatoire pour passer un niveau. Il n'y a au maximum qu'un seul cadenas et qu'une seule clef par niveau. La clé ne peut pas être sur une case piège à ours. Le cadenas non plus.

1.1.3 Principales différences avec le Sokoban

Bien que Helltaker soit de la famille du Sokoban, il est possible d'observer plusieurs différences entre les deux jeux. Tout d'abord les **buts des deux jeux** sont différents : alors qu'il s'agit de placer des objets sur certaines cases précises de la grille pour le Sokoban, dans Helltaker le but est d'atteindre une case précise. De plus, le **nombre d'actions** est illimité dans le Sokoban, tandis qu'un compteur empêche le joueur d'effectuer plus qu'un certain nombre d'actions. Le **coût des actions** est également une nouveauté de Helltaker en comparaison du Sokoban : toutes les actions ont un coût et certaines sont plus coûteuses que les autres. Dans le cas du Sokoban, l'action de **pousser un objet** fait également déplacer le personnage alors que Helltaker reste immobile lorsqu'il pousse un rocher ou un squelette.

Pour finir, la principale différence entre le Sokoban et Helltaker est que ce dernier jeu introduit de **nouveaux types de cases** :

- les squelettes, qui ressemblent aux caisses du sokoban et aux rochers de Helltaker mais qui peuvent être détruits
- les cases cadenas et clefs
- les pièges à ours, qui peuvent changer d'état

1.2 Le problème en STRIPS

Les règles étant désormais précisées, nous modéliserons le problème en STRIPS. Il est possible de retrouver la modélisation complète dans le dossier Modelisation du rendu, sous le nom strips.md. En effet, pour plus de clarté et de lisibilité, nous détaillerons ci-dessous uniquement les principales étapes de modélisation et les conséquences des règles du jeu sur ces dernières.

1.2.1 Les prédicats

Prédicat	Fluent ?
wall(X,Y)	Non
helltaker(X,Y)	**Oui**
demon(X,Y)	Non
rock(X,Y)	**Oui**
skeleton(X,Y)	**Oui**
counter(X)	**Oui**
lock(X,Y,S)	**Oui**
key(X,Y)	Non
openTrap(X,Y)	Non
trap(X,Y,S)	**Oui**
plusOne(X,X')	Non

1.2.2 But et initialisation

Le but est d'atteindre une case adjacente à la case démone. Il y a donc plusieurs possibilités pour gagner une partie : se positionner sur la case au-dessus, en-dessous, directement à droit ou à gauche de la démone. Nous pouvons donc le modéliser comme ceci :

But(demon(X,Y),
plusOne(Y,H), plusOne(B,Y), plusOne(X,D), plusOne(G,X),
(helltaker(X,H) | helltaker(X,B) | helltaker(D,Y) | helltaker(G,Y)))

Nous avons utilisé le **niveau 6** de Helltaker comme modèle pour l'initialisation du problème et la modélisation des actions. Seul l'emplacement des objets sur la grille devrait changer d'un niveau à l'autre, ainsi que le nombre de piège à ours changeant d'état. Nous considérons la grille de jeu comme un repère cartésien : le X désigne la position en abscisse et le Y celle en ordonné. Pour initialiser le niveau, nous avons tout d'abord placé les murs, la démone, les deux pièges à ours, le cadenas et la clef qui sont des éléments qui ne se déplace pas. Nous avons ensuite placé la position initiale du joueur Helltaker, des rochers et des squelettes qui seront amenés par la suite à se déplacer.

Pour finir l'initialisation, nous avons défini le prédicat plusOne d'artité deux. Il nous permettra de faire des incrémentations et des décrémentations par pas de un. Le compteur de ce niveau commençant à 43, il s'agit de la plus haute valeur à décrémenter. Ce même compteur s'arrête à -1 lorsque la partie est perdue : il s'agit de la plus petite valeur à prendre en compte. Ainsi ce prédicat nous servira à gérer le compteur mais aussi à nous déplacer sur la grille.

1.2.3 Les actions

Pour chaque action, une précondition vérifie que le compteur n'est pas en dessous de zéro et que le coup est donc possible. De plus, chaque effet des actions implique une décrémentation de un du compteur sauf pour les actions durant lesquelles Helltaker passent ou restent sur un piège à l'état dangereux. Dans ce cas, le compteur est décrémenté de deux.

Tout d'abord, nous avons modélisé les **déplacements simples**, c'est-à-dire ceux sur la case vide et sur les pièges à l'état innofensif. Quatre actions y sont dédiées : une pour chaque direction possible, c'est-à-dire haut, bas, droite et gauche. La case sur laquelle se dirige le personnage ne doit être ni un mur, ni la démone, ni un cadenas fermé, ni la clef ou un piège qui s'ouvrirait au cours de cette action.

Ensuite, il a fallu gérer le **déplacement d'objets**. Huit actions ont été nécessaires : une pour chaque objet déplaçable dans les quatre directions. Il a cette fois-ci fallu faire attention à ce qu'Helltaker soit bien adjacent à un squelette ou à un rocher et que la case suivant n'empêche pas le déplacement de l'objet. Huit autres actions ont été créées pour l'élimination des squelettes car ils peuvent être éliminés en étant poussé contre un mur ou contre un rocher et ce dans les quatre directions.

Concernant la **gestion de la clef et du cadenas**, seulement quatre actions ont dû être implémentées. En effet, Helltaker doit se déplacer sur une case clef pour dévérouiller un cadenas. Ainsi les préconditions incluent le fait d'être adjacent à une clef. Les effets impliquent que le troisième paramètre du prédicat lock concernant l'unique cadenas du niveau change à 1, indiquant ainsi que le cadenas est ouvert. Bien sûr, il a fallu réaliser une action pour chaque direction.

Nous devons ensuite modéliser les **déplacements sur les pièges à ours**, ceux qui restent ouverts tout au long du niveau et ceux qui sont ouverts à l'instant ou Helltaker se déplace sur eux. Le niveau six ne contient que des pièges qui changent d'états, cependant, afin d'être le plus généraliste possible, nous avons tenu à prendre en compte l'autre type de piège. Le changement principale ici est la décrémentation du compteur qui n'est pas de un mais de deux. De plus, lorsque Helltaker est sur un piège à ours et qu'il pousse un objet ou tue un ennemi, il restera sur cette case. Si le piège n'est pas dans un état inoffensif, le compteur devra se décrémenter de deux points en conséquence.

La gestion des pièges qui s'ouvrent et se ferment se termine avec l'intégration d'une précondition permettant d'ajouter aux effets **le bon changement d'état**. Il a fallu le double d'actions : une action à réaliser lorsque les pièges sont ouverts pour pouvoir les fermer dans les effets, et une autre à réaliser lorsqu'ils sont fermés pour pouvoir ensuite les ouvrir. Les seules actions que nous n'avons pas besoin de modéliser deux fois pour changer l'état des pièges sont celles qui impliquent de rester sur un piège qui va s'ouvrir. En effet, ces actions s'appliquent uniquement lorsque Helltaker est déjà sur un piège fermé et qui s'ouvrira donc à sa prochaine action. Cette condition rend inutile la modélisation des deux cas d'état des pièges.

Nous terminons ainsi la modélisation STRIPS avec un total de **100 actions**.

2 Python

Dans cette deuxième partie, nous aborderons la partie «implémentation» avec Python pour la recherche dans un espace d'état. Tout en s'inspirant de la modélisation STRIPS.

2.1 Représentation du problème

H	D	L	#	B	M	K	S	T
hero	goals	lock	walls	rocks	skeletons	key	openTraps	slidingTrapsF

U	O	P	Q
slidingTrapsT	openTrapsrocks	slidingTrapsFrocks	slidingTrapsTrocks

Avec slidingTrapsF et slidingTrapsT les pièges qui peuvent changer d'état et qui sont respectivement à l'état fermé et ouvert au moment t.

Cette représentation nous permet donc de faire les divisions suivantes :

- slidingTrapsTrocks(rock sur piège ouvert) en slidingTrapsT + rocks
- slidingTrapsFrocks(rock sur piège fermé) en slidingTrapsF + rocks
- openTrapsrocks(rock sur piège permanemment ouvert) en openTraps + rocks

Les slidingTraps (slidingTrapsF,slidingTrapsT) seront représentés par un tuple de 3 éléments (positionX,positionY,bool) avec bool un booléen mis à True lorsque le piège est ouvert (slidingTrapsT) et False lorsqu'il est fermé (slidingTrapsF).

A la fin nous obtenons le modèle suivant pour nos états(fluents) et la map_rules(caractéristiques non-fluents) :

Etat	hero	rocks	skeletons	counter	lock	slidingTraps
Format	(x,y)	{(x',y')...}	{(x1,y1)...}	n	{(x2,y2,bool)...}	{(x3,y3,bool)...}

Le booléen de **lock** est True si la clé a déjà été récupérée par le héro(cad lorsqu'il va passé sur la case où est située la clé).

map_rules	goals	walls	key	openTraps	dim
Format	{(x,y)...}	{(x',y')...}	(x2,y2)	{(x2,y2,bool)...}	m

2.2 Choix d'implémentation et structures de données

Pour ce qu'il en est de notre **choix d'implémentation** en Python, nous avons opté pour la programmation fonctionnelle plutôt que la programmation orientée objet. En effet, ce choix nous paraissait le plus facile à mettre en œvre surtout au regard de la hashabilité de nos données.

De plus, dans cette implémentation nous avons utilisé des fonctions et non des procédures pour ainsi réduire les risques de réaliser des effets de bords. Egalement le fait

d'utiliser des procédures nous auraient probablement causé des problèmes liées à l'irréversibilité quant à la modification de nos états (par exemple pour retrouver facilement l'état père d'un autre état sans devoir le modifier) d'autant plus que dans notre cas, nous désirerions par exemple pour un état donné, renvoyer son successeur sans pour autant modifier l'état courant.

Pour la recherche d'état, nous avons procédé par un **parcours en largeur d'abord avec une sauvegarde des états visités**(pour éviter les boucles).

Pour améliorer la recherche, nous avons utiliser la **distance de Manhattan** comme **heuristique**, vu qu'elle est minorante. Ainsi nous traiterons les états successeurs par ordre croissant selon l'heuristique additionnée au compteur du successeur. Et pour un meilleur filtrage, nous ne retenons que les successeurs pour lesquels le compteur est au moins égal au nombre de case pour atteindre un but(ici la distance de Manhattan).

Quant aux **structures de données** utilisées, nous avons utilisé :

- des **conteneurs (collections.namedtuple())** pour stocker les états et les actions.
- des **dictionnaires** (avec des clés hashables) pour stocker la map_rules qui contient toutes les caractéristiques non-fluentes relatifs à chaque map. Ce qui nous permet d'obtenir plus de lisibilité, de clarté dans notre code.

2.3 Expérimentations pratiques

Ci-dessous, les tests effectués sur les différentes cartes ainsi que le temps d'exécution moyen (qui varie grandement en fonctions des machines).

- **MAP TESTS**

maps	locknkey	corridor	traps1	spikes1	spikes2	spikes3
plans	gggddddddddd	bbb	None	dbddh	dddddd	ddddddd

maps	spikes4	trapsmob1	trapsmob2	trapsTTT	trapsUUU
plans	bbb	ddddd	ddddd	dddd	ddddd

- **MAP LEVELS**

maps	**plans**	**temps(s)**
level1	bgbbgggggbgbbddhhddddbd	0.13
level2	dhhhhhhdddbddbbbbggb	0.025
level3	gggggbbbbgghbddddddddhhhhhh	0.17
level4	bbbdbbdddhgghhdbbdddb	2.4
level5	bbbbddhdddbhghhgghhhh	0.43
level6	gbddbbggbggbbddddhhbggbbbddhhddddbbggbb	8.5
level7	hhbgghhdbbbggghhhhddhbddddhhddhh	4.3
level8	hghhhhhhhhdd	0.006
level9	dhhddddhbddghhddddbbddhhdgggghhhgh	310

3 ASPPLAN

Ensuite nous avons choisi d'aborder le problème au moyen d'ASP, en s'inspirant de la modélisation STRIPS et avec des similitudes avec la partie en Python.

3.1 Représentation du problème

H	D	L	#	B	M	K	S - T - U
at	demon	lock	wall	box	mob	key	trap
Fluent	Non Fluent	Fluent	Non Fluent	Non Fluent	Fluent	Not Fluent	Fluent

Tous les éléments prennent au minimum 2 arguments : Leurs coordonnées dans le plan symbolisées par X et Y.

Il y a toutefois quelques particularités pour les traps. Au départ, nous avions choisi d'utiliser 2 types de traps : les statiques (spikes) et les dynamiques. Finalement, pour diminuer le nombre de règles et de conditions, nous utilisons qu'un seul type de trap, qui prend en 3ème argument un entier, soit -1, soit 0, soit 1.

Ainsi 0 correspond à un trap statique (les spikes), -1 à un trap False et 1 à un trap True.

En utilisant ce système, on peut multiplier ce 3ème argument par -1 à chaque action du joueur pour permuter les traps dynamiques. Comme le trap statique vaut 0 alors le multiplier ne changera pas la valeur.

Pour savoir si un joueur possède la clé, on utilise tout simplement une variable possede_cle qui devient vraie lorsque le joueur marche sur la case où se trouve la clé.

3.2 Choix d'implémentation et structures de données

Pour l'implémentation on passe par un programme en python dont le but va être de générer le fichier ASP, et de le donner en entrée au solveur clingo dont l'intégration se fait très bien en python grâce à la librairie éponyme.

On regarde donc chaque case des fichiers .txt des niveaux et on rajoute les informations sur les éléments dans une variable string.

Ensuite on y rajoute toutes les règles et conditions contenues dans le fichier ASP et on utilise les fonctions de clingo pour obtenir la solution.

3.3 Expérimentations pratiques

Lors des expérimentations pratiques, nous nous sommes rendus compte de plusieurs soucis :

- De temps en temps, la solution renvoyée ne donne pas les actions dans le bon ordre. En suivant l'ordre de l'incrément associé à chaque action, on s'aperçoit que la solution est correcte mais si on suit l'ordre dans lequel les actions apparaissent alors c'est erroné.

  Pour palier ça, nous avons inclut un ordonnancement dans la fonction d'affichage (affichage2) du fichier **helltaker_utils.py**.

- Le programme "comprend" lorsqu'une action a un coût de 2 et le prend en compte dans la solution donnée, cependant nous n'avons pas réussi à faire en sorte que l'incrément augmente de 2. Par conséquent, il peut y avoir un résidus "d'actions" inutiles à la fin.

  En principe, la seule action possible devrait être "nop" mais de temps en temps, on obtient une action normalement impossible comme "top" alors qu'il y a un mur au dessus.

- Enfin, de temps en temps, le programme ne trouve pas de solutions sans qu'on comprenne d'où vient le problème.

- **MAP LEVELS**

maps	**plans**	**temps(s)**
level1	bgggggbbgbgbbddhhddddbd	1.09
level2	Ne donne aucune solution	//
level3	Ne donne aucune solution	//
level4	bbbdbbdddhgghhhdbbddddb	1.46
level5	bbbbbddddddbhhhhhggghhdh	3.62
level6	Ne donne aucune solution	//
level7	Ne donne aucune solution	//
level8	hdhhhhhhhhgg	0.19
level9	dhhddddhbddghhddddbbddhhdgggghhhgh	12.23

4 Comparaison

Après avoir implémenté ces méthodes pour répondre à notre problème, nous avons cherché à les comparer pour en déduire la plus performante. Nous avons essayé de faire une analyse complète en prenant en compte la difficulté d'implémentation, le temps d'exécution et le taux d'erreurs renvoyés par nos algorithmes. Nous n'avons pas pris en compte la place mémoire sollicité mais il aurait pu être intéressant d'en faire l'analyse.

4.1 Difficulté d'implémentation des méthodes

Selon nous, la recherche dans un espace d'état nous a paru plus facile et donc rapide à implémenter. Tout d'abord parce qu'il s'agit d'un langage de programmation assez généraliste qui propose de nombreuses structures de données. Nous sommes donc plus libre dans notre code. De plus il s'agit d'un langage fonctionnel ce qui rend le code plus précis, concis, facilement testable et facilement vérifiable.

Pour ce qui est de l'ASP, la première difficulté a été une difficulté d'adaptation. En effet, ce n'est pas un langage de programmation aussi intuitif que Python et nous y sommes moins habitué. Toutefois, une fois cette étape franchie, il s'avère simple de paramétrer toutes les données dans notre problème. La difficulté vient alors du nombre très important de règles et donc de la probabilité non négligeable de faire des erreurs sans s'en rendre compte. A titre d'exemple, notre fichier ASP fait environ 550 lignes et toutes les règles sont similaires à d'autres. On peut donc facilement s'embrouiller. Enfin la dernière difficulté vient de la gestion des erreurs. Lorsque la sortie indique "Insatisfiable", il est très compliqué de savoir pour quelle raison est-ce que aucun modèle n'a été trouvé car on ne peut pas avoir un suivi précis des variables ou des étapes de recherche comme on pourrait avoir dans un langage de programmtion fonctionnel grâce à des "print".

4.2 Temps d'exécution

Concernant le temps d'exécution, le programme python prend un temps inférieur à une seconde en moyenne pour trouver une solution. Cependant, le niveau neuf a un temps particulièrement long du fait de sa difficulté : il est jonché de nombreux rochers.

Le temps d'exécution est largement plus long long sur certains niveaux à cause de la condition servant à vérifier si un état a déjà été parcouru ou non. Cette condition n'est pas toujours optimale en fonction des niveaux. Par exemple, c'est le cas pour la grille numéro deux. Il a donc fallu implémenter une autre fonction de recherche spécifique à ce niveau qui modifie cette condition pour permettre au temps d'exécution de descendre sous une seconde. Cependant, cette nouvelle fonction est peu performante sur les autres niveaux : vingt secondes au moins sont alors nécessaires à l'exécution.

Contrairement à la recherche dans un espace d'états, l'ASP a tendance a prendre plus de temps sur la plupart des niveaux, ce qui semble contre-intuitif. Ceci peut bien sûr venir du fait que les deux méthodes n'ont pas été testées sur les mêmes machines ou que l'ASP, ici, n'utilise pas d'heuristique. En revanche, dans le cas du 9eme niveau l'ASP trouve une solution en seulement 12 secondes contre 310 secondes pour la recherche d'états en python. On peut donc en déduire que l'ASP devient extrêmement intéressant pour les niveaux plus complexes, là où la recherche d'état est en moyenne plus rapide pour des niveaux simples ou moyens.

4.3 Taux d'erreurs

En ce qui concerne le taux d'erreurs, la méthode de recherche dans un espace d'état renvoie toujours un plan valide et permettant de remplir le niveau avec succès, peu importe le niveau du jeu. La méthode ASP ne nous renvoie malheureusement pas toujours de solution. Nous n'avons pas eu le temps de comprendre pourquoi. Le taux d'échec sur le jeu Helltaker s'élève environ à 4 niveaux sur neuf, soit un peu moins de 50%.

5 Conclusion

Ainsi, en réalisant ce projet, nous avons pu chercher des solutions pour résoudre les niveaux puzzle du jeu vidéo Helltaker, dont le principe est assez similaire au Sokoban comme nous avons pu le remarquer précédemment.

En éprouvant les possibilités et les actions possibles dans le jeu, nous avons pu en déduire ses règles pour en faire une liste claire et précise. Nous nous sommes ensuite appuyés sur cette dernière afin de modéliser le problème en STRIPS. Grâce aux notions et aux compétences acquises en IA02, nous avons pu implémenter deux méthodes permettant de trouver une solution au problème : la recherche dans un espace d'états en python, et l'ASP.

Dans l'ensemble, nous sommes plutôt satisfait de nos résultats car chacune de ces méthodes abouties non seulement à des plans valides mais également à une solution aux niveaux du jeu. Nous avons tenté d'optimiser au maximum le temps d'exécution de nos programmes et de prendre garde à la mémoire utilisée pour la recherche de plans.

⋆ ⋆ ⋆

6 Annexes

A. Modélisation STRIPS

Nous nous servirons du niveau 6 de Helltaker comme modèle pour l'initialisation du problème. Seul l'emplacement des objets sur la grille devrait changer d'un niveau à l'autre. **Une version plus lisible de la modélisation STRIPS se trouve dans le dossier modélisation du projet.**

```
Init(
    demon(6,1),
    helltaker(3,8),
    counter(43),
    lock(5,2,0),
    key(4,6),
    trap(2,5,0),trap(3,5,0),
    skeleton(2,4), skeleton(6,3),
    rock(6,2), rock(4,3), rock(4,4), rock(5,4), rock(3,5), rock(2,7), rock(3,7),
    rock(4,7),
    wall(5,0), wall(6,0), wall(4,1), wall(7,1), wall(1,2), wall(2,2), wall(3,2),
    wall(4,2), wall(8,2), wall(1,3), wall(7,3), wall(1,4), wall(3,4), wall(8,4),
    wall(1,5), wall(6,5), wall(7,5), wall(0,6), wall(5,6), wall(6,6), wall(1,7),
    wall(5,7), wall(1,8), wall(5,8), wall(2,9), wall(3,9), wall(4,9),
    plusOne(-1,0),
    plusOne(0,1), plusOne(1,2), plusOne(2,3), plusOne(3,4), plusOne(4,5),
    plusOne(5,6), plusOne(6,7), plusOne(7,8), plusOne(8,9), plusOne(9,10),
    plusOne(10,11), plusOne(11,12), plusOne(12,13), plusOne(13,14), plusOne(14,15),
    plusOne(15,16), plusOne(16,17), plusOne(17,18), plusOne(18,19), plusOne(19,20),
    plusOne(20,21), plusOne(21,22), plusOne(22,23), plusOne(23,24), plusOne(24,25),
    plusOne(25,26), plusOne(26,27), plusOne(27,28), plusOne(28,29), plusOne(29,30),
    plusOne(30,31), plusOne(31,32), plusOne(32,33), plusOne(33,34), plusOne(34,35),
    plusOne(35,36), plusOne(36,37), plusOne(37,38), plusOne(38,39), plusOne(39,40),
    plusOne(40,41), plusOne(41,42), plusOne(42,43)
)
```

Le but est d'atteindre une case adjacente à la case démone. Nous pouvons le modéliser comme ceci :

```
But(
    demon(X,Y),
    plusOne(Y,H), plusOne(B,Y), plusOne(X,D), plusOne(G,X),
    (helltaker(X,H) | helltaker(X,B) | helltaker(D,Y) | helltaker(G,Y))
)
```

goUp(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y), counter(C),trap(TX,TY,0),trap(UX,UY,0), plusOne(Y,Y'),-wall(X,Y'),-demon(X,Y'),-rock(X,Y'),-skeleton(X,Y'), -lock(X,Y',0),-key(X,Y'),-openTrap(X,Y'),-trap(X,Y',0), -counter(-1)

Effect : plusOne(Y,Y'), helltaker(X,Y'), -helltaker(X,Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

goDown(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y), counter(C),trap(TX,TY,0),trap(UX,UY,0), plusOne(Y',Y),-wall(X,Y'),-demon(X,Y'),-rock(X,Y'),-skeleton(X,Y'), -lock(X,Y',0),-key(X,Y'),-openTrap(X,Y'),-trap(X,Y',0), -counter(-1)

Effect : plusOne(Y',Y), helltaker(X,Y'), -helltaker(X,Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

goLeft(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y), counter(C),trap(TX,TY,0),trap(UX,UY,0), plusOne(X',X),-wall(X',Y),-demon(X',Y),-rock(X',Y),-skeleton(X',Y), -lock(X',Y,0),-key(X',Y),-openTrap(X',Y),-trap(X',Y,0), -counter(-1)

Effect : plusOne(X',X), helltaker(X',Y), -helltaker(X,Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

goRight(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y), counter(C),trap(TX,TY,0),trap(UX,UY,0), plusOne(X,X'),-wall(X',Y),-demon(X',Y),-rock(X',Y),-skeleton(X',Y), -lock(X',Y,0),-key(X',Y),-openTrap(X',Y),-trap(X',Y,0), -counter(-1)

Effect : plusOne(X,X'), helltaker(X',Y), -helltaker(X,Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

Ensuite nous modélisons le déplacement des objets, c'est-à-dire des rochers et des squelettes. Ainsi il faut vérifier que la case adjacente à Helltaker correspond bien à un objet et que la case suivante à l'objet ne soit ni un autre objet ni un mur. Le personnage ne bouge pas lors de ce type d'actions.

pushRockUp(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(Y,Y'), rock(X,Y'), plusOne(Y',Y"), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0) **Effect :** plusOne(Y,Y'), plusOne(Y',Y"), rock(X,Y"), -rock(X,Y'), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushRockDown(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(Y',Y), rock(X,Y'), plusOne(Y",Y'), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y',Y), plusOne(Y",Y'), rock(X,Y"), -rock(X,Y'), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushRockLeft(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(X',X), rock(X',Y), plusOne(X",X'), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X',X), plusOne(X",X'), rock(X",Y), -rock(X',Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushRockRight(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(X,X'), rock(X',Y), plusOne(X',X"), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X,X'), plusOne(X',X"), rock(X",Y), -rock(X',Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushSkeletonUp(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(Y,Y'), skeleton(X,Y'), plusOne(Y',Y"), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y,Y'), plusOne(Y',Y"), skeleton(X,Y"), -skeleton(X,Y'), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushSkeletonDown(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(Y',Y), skeleton(X,Y'), plusOne(Y",Y'), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y',Y), plusOne(Y",Y'), skeleton(X,Y"), -skeleton(X,Y'), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushSkeletonLeft(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(X',X), skeleton(X',Y), plusOne(X",X'), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X',X), plusOne(X",X'), skeleton(X",Y), -skeleton(X',Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushSkeletonRight(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(X,X'), skeleton(X',Y), plusOne(X',X"), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X,X'), plusOne(X',X"), skeleton(X",Y), -skeleton(X',Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

Les actions qui suivent modélisent l'élimination des ennemis : il s'agit également de pousser un squelette, mais cette fois ci un mur ou un rocher est adjacent au squelette ce qui aura pour effet de le faire disparaitre et non pas de le déplacer.

wallKillUp(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(Y,Y'), skeleton(X,Y'), plusOne(Y',Y"), wall(X,Y")

Effect : plusOne(Y,Y'), -skeleton(X,Y'), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

rockKillUp(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(Y,Y'), skeleton(X,Y'), plusOne(Y',Y"), rock(X,Y")

Effect : plusOne(Y,Y'), -skeleton(X,Y'), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

wallKillDown(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(Y',Y), skeleton(X,Y'), plusOne(Y",Y'), wall(X,Y")

Effect : plusOne(Y',Y), -skeleton(X,Y'), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

rockKillDown(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(Y',Y), skeleton(X,Y'), plusOne(Y",Y'), rock(X,Y")

Effect : plusOne(Y',Y), -skeleton(X,Y'), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

wallKillLeft(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(X',X), skeleton(X',Y), plusOne(X",X'), wall(X",Y)

Effect : plusOne(X',X), -skeleton(X',Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

rockKillLeft(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(X',X), skeleton(X',Y), plusOne(X",X'), rock(X",Y)

Effect : plusOne(X',X), -skeleton(X',Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

wallKillRight(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(X,X'), skeleton(X',Y), plusOne(X',X"), wall(X",Y)

Effect : plusOne(X,X'), -skeleton(X',Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

rockKillRight(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(X,X'), skeleton(X',Y), plusOne(X',X"), rock(X",Y)

Effect : plusOne(X,X'), -skeleton(X',Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

Pour la gestion des clefs et des cadenas, nous avons considéré qu'une seule clef et qu'un seul cadenas au maximum ne pouvait exister par niveau. Helltaker doit utiliser ces actions pour passer sur une case clef, ce qui fera passer le cadenas de l'état fermé (0) à ouvert (1).

obtainKeyUp(X,Y,A,B,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), lock(A,B,0), counter(C), -counter(-1), plusOne(Y,Y'), key(X,Y')

Effect : plusOne(Y,Y'), helltaker(X,Y'), -helltaker(X,Y), lock(A,B,1), -lock(A,B,0), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

obtainKeyDown(X,Y,A,B,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), lock(A,B,0), counter(C), -counter(-1), plusOne(Y',Y), key(X,Y')

Effect : plusOne(Y',Y), helltaker(X,Y'), -helltaker(X,Y), lock(A,B,1), -lock(A,B,0), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

obtainKeyLeft(X,Y,A,B,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), lock(A,B,0), counter(C), -counter(-1), plusOne(X',X), key(X',Y)

Effect : plusOne(X',X), helltaker(X',Y), -helltaker(X,Y), lock(A,B,1), -lock(A,B,0), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

obtainKeyRight(X,Y,A,B,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), lock(A,B,0), counter(C), -counter(-1), plusOne(X,X'), key(X',Y)

Effect : plusOne(X,X'), helltaker(X',Y), -helltaker(X,Y), lock(A,B,1), -lock(A,B,0), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

Nous devons ensuite modéliser les déplacements sur les pièges à ours, ceux qui restent ouverts tout au long du niveau et ceux qui sont ouverts à l'instant ou Helltaker se déplace sur eux. Le compteur se décrémente ici de deux.

goUpOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y,Y'), -rock(X,Y'), -skeleton(X,Y'), openTrap(X,Y')

Effect : plusOne(Y,Y'), helltaker(X,Y'), -helltaker(X,Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

goDownOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y',Y), -rock(X,Y'), -skeleton(X,Y'), openTrap(X,Y')

Effect : plusOne(Y',Y), helltaker(X,Y'), -helltaker(X,Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

goLeftOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X',X), -rock(X',Y), -skeleton(X',Y), openTrap(X',Y)

Effect : plusOne(X',X), helltaker(X',Y), -helltaker(X,Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

goRightOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X,X'), -rock(X',Y), -skeleton(X',Y), openTrap(X',Y)

Effect : plusOne(X,X'), helltaker(X',Y), -helltaker(X,Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

goUpOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y,Y'), -rock(X,Y'), -skeleton(X,Y'), trap(X,Y',0)

Effect : plusOne(Y,Y'), helltaker(X,Y'), -helltaker(X,Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

goDownOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y',Y), -rock(X,Y'), -skeleton(X,Y'), trap(X,Y',0)

Effect : plusOne(Y',Y), helltaker(X,Y'), -helltaker(X,Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

goLeftOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X',X), -rock(X',Y), -skeleton(X',Y), trap(X',Y,0)

Effect : plusOne(X',X), helltaker(X',Y), -helltaker(X,Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

goRightOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X,X'), -rock(X',Y), -skeleton(X',Y), trap(X',Y,0)

Effect : plusOne(X,X'), helltaker(X',Y), -helltaker(X,Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

De plus, lorsque Helltaker est sur un piège à ours actif et qu'il pousse un objet ou tue un ennemi, il restera sur cette case. Le compteur devra donc décrémenter de deux points en conséquence.

pushRockUpWhileOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y,Y'), rock(X,Y'), plusOne(Y',Y"), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y,Y'), plusOne(Y',Y"), rock(X,Y"), -rock(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushRockDownWhileOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y',Y), rock(X,Y'), plusOne(Y",Y'), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y',Y), plusOne(Y",Y'), rock(X,Y"), -rock(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushRockLeftWhileOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X',X), rock(X',Y), plusOne(X",X'), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X',X), plusOne(X",X'), rock(X",Y), -rock(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushRockRightWhileOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X,X'), rock(X',Y), plusOne(X',X"), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X,X'), plusOne(X',X"), rock(X",Y), -rock(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushSkeletonUpWhileOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y,Y'), skeleton(X,Y'), plusOne(Y',Y"), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y,Y'), plusOne(Y',Y"), skeleton(X,Y"), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushSkeletonDownWhileOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y',Y), skeleton(X,Y'), plusOne(Y",Y'), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y',Y), plusOne(Y",Y'), skeleton(X,Y"), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushSkeletonLeftWhileOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X',X), skeleton(X',Y), plusOne(X",X'), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X',X), plusOne(X",X'), skeleton(X",Y), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushSkeletonRightWhileOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X,X'), skeleton(X',Y), plusOne(X',X"), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X,X'), plusOne(X',X"), skeleton(X",Y), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

wallKillUpWhileOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y,Y'), skeleton(X,Y'), plusOne(Y',Y"), wall(X,Y")

Effect : plusOne(Y,Y'), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

rockKillUpWhileOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y,Y'), skeleton(X,Y'), plusOne(Y',Y"), rock(X,Y")

Effect : plusOne(Y,Y'), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

wallKillDownWhileOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y',Y), skeleton(X,Y'), plusOne(Y",Y'), wall(X,Y")

Effect : plusOne(Y',Y), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

rockKillDownWhileOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y',Y), skeleton(X,Y'), plusOne(Y",Y'), rock(X,Y")

Effect : plusOne(Y',Y), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

wallKillLeftWhileOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X',X), skeleton(X',Y), plusOne(X",X'), wall(X",Y)

Effect : plusOne(X',X), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

rockKillLeftWhileOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X',X), skeleton(X',Y), plusOne(X",X'), rock(X",Y)

Effect : plusOne(X',X), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

wallKillRightWhileOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X,X'), skeleton(X',Y), plusOne(X',X"), wall(X",Y)

Effect : plusOne(X,X'), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

rockKillRightWhileOnOpenTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X,X'), skeleton(X',Y), plusOne(X',X"), rock(X",Y)

Effect : plusOne(X,X'), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushRockUpWhileOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), trap(X,Y,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y,Y'), rock(X,Y'), plusOne(Y',Y"), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y,Y'), plusOne(Y',Y"), rock(X,Y"), -rock(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushRockDownWhileOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), trap(X,Y,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y',Y), rock(X,Y'), plusOne(Y",Y'), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y',Y), plusOne(Y",Y'), rock(X,Y"), -rock(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushRockLeftWhileOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), trap(X,Y,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X',X), rock(X',Y), plusOne(X",X'), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X',X), plusOne(X",X'), rock(X",Y), -rock(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushRockRightWhileOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), trap(X,Y,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X,X'), rock(X',Y), plusOne(X',X"), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X,X'), plusOne(X',X"), rock(X",Y), -rock(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushSkeletonUpWhileOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), trap(X,Y,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y,Y'), skeleton(X,Y'), plusOne(Y',Y"), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y,Y'), plusOne(Y',Y"), skeleton(X,Y"), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushSkeletonDownWhileOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), trap(X,Y,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y',Y), skeleton(X,Y'), plusOne(Y",Y'), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y',Y), plusOne(Y",Y'), skeleton(X,Y"), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushSkeletonLeftWhileOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), trap(X,Y,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X',X), skeleton(X',Y), plusOne(X",X'), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X',X), plusOne(X",X'), skeleton(X",Y), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

pushSkeletonRightWhileOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), trap(X,Y,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X,X'), skeleton(X',Y), plusOne(X',X"), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X,X'), plusOne(X',X"), skeleton(X",Y), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

wallKillUpWhileOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), trap(X,Y,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y,Y'), skeleton(X,Y'), plusOne(Y',Y"), wall(X,Y")

Effect : plusOne(Y,Y'), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

rockKillUpWhileOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), trap(X,Y,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y,Y'), skeleton(X,Y'), plusOne(Y',Y"), rock(X,Y")

Effect : plusOne(Y,Y'), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

wallKillDownWhileOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), trap(X,Y,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y',Y), skeleton(X,Y'), plusOne(Y",Y'), wall(X,Y")

Effect : plusOne(Y',Y), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

rockKillDownWhileOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), trap(X,Y,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y',Y), skeleton(X,Y'), plusOne(Y",Y'),

rock(X,Y")

Effect : plusOne(Y',Y), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

wallKillLeftWhileOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), trap(X,Y,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X',X), skeleton(X',Y), plusOne(X",X'), wall(X",Y)

Effect : plusOne(X',X), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

rockKillLeftWhileOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), trap(X,Y,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X',X), skeleton(X',Y), plusOne(X",X'), rock(X",Y)

Effect : plusOne(X',X), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

wallKillRightWhileOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), trap(X,Y,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X,X'), skeleton(X',Y), plusOne(X',X"), wall(X",Y)

Effect : plusOne(X,X'), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

rockKillRightWhileOnTrap(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,0),trap(UX,UY,0), trap(X,Y,0), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X,X'), skeleton(X',Y), plusOne(X',X"), rock(X",Y)

Effect : plusOne(X,X'), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,0), -trap(UX,UY,0), trap(TX,TY,1), trap(UX,UY,1)

Pour terminer la gestion des pièges qui s'ouvrent et se ferment, il a fallu intégrer une nouvelle précondition pour pouvoir ajouter aux effets le bon changement d'état. C'est pourquoi il a fallu le double d'actions : une action à réaliser lorsque les pièges sont ouverts pour pouvoir les fermer dans les effets, et une autre à réaliser lorsqu'ils sont fermés pour pouvoir ensuite les ouvrir.

Les seules actions que nous n'avons pas besoin de modéliser deux fois pour changer l'état des pièges sont celles qui impliquent de rester sur un piège qui va s'ouvrir. En effet, ces actions ne s'appliquent uniquement lorsque Helltaker est déjà sur un piège fermé et qui s'ouvrira donc à sa prochaine action. Cette condition rend inutile la modélisation des deux cas d'état des pièges.

goUpWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y), counter(C),trap(TX,TY,1),trap(UX,UY,1), plusOne(Y,Y'),-wall(X,Y'),-demon(X,Y'),-rock(X,Y'),-skeleton(X,Y'),-lock(X,Y',0),-key(X,Y'),-openTrap(X,Y'),-trap(X,Y',0), -counter(-1)

Effect : plusOne(Y,Y'), helltaker(X,Y'), -helltaker(X,Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

goDownWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y), counter(C),trap(TX,TY,1),trap(UX,UY,1), plusOne(Y',Y),-wall(X,Y'),-demon(X,Y'),-rock(X,Y'),-skeleton(X,Y'),-lock(X,Y',0),-key(X,Y'),-openTrap(X,Y'),-trap(X,Y',0), -counter(-1)

Effect : plusOne(Y',Y), helltaker(X,Y'), -helltaker(X,Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

goLeftWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y), counter(C),trap(TX,TY,1),trap(UX,UY,1), plusOne(X',X),-wall(X',Y),-demon(X',Y),-rock(X',Y),-skeleton(X',Y),-lock(X',Y,0),-key(X',Y),-openTrap(X',Y),-trap(X',Y,0), -counter(-1)

Effect : plusOne(X',X), helltaker(X',Y), -helltaker(X,Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

goRightWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y), counter(C),trap(TX,TY,1),trap(UX,UY,1), plusOne(X,X'),-wall(X',Y),-demon(X',Y),-rock(X',Y),-skeleton(X',Y),-lock(X',Y,0),-key(X',Y),-openTrap(X',Y),-trap(X',Y,0), -counter(-1)

Effect : plusOne(X,X'), helltaker(X',Y), -helltaker(X,Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

pushRockUpWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(Y,Y'), rock(X,Y'), plusOne(Y',Y"), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y,Y'), plusOne(Y',Y"), rock(X,Y"), -rock(X,Y'), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

pushRockDownWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(Y',Y), rock(X,Y'), plusOne(Y",Y'), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y',Y), plusOne(Y",Y'), rock(X,Y"), -rock(X,Y'), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

pushRockLeftWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(X',X), rock(X',Y), plusOne(X",X'), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X',X), plusOne(X",X'), rock(X",Y), -rock(X',Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

pushRockRightWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(X,X'), rock(X',Y), plusOne(X',X"), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X,X'), plusOne(X',X"), rock(X",Y), -rock(X',Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

pushSkeletonUpWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(Y,Y'), skeleton(X,Y'), plusOne(Y',Y"), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y,Y'), plusOne(Y',Y"), skeleton(X,Y"), -skeleton(X,Y'), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

pushSkeletonDownWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(Y',Y), skeleton(X,Y'), plusOne(Y",Y'), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y',Y), plusOne(Y",Y'), skeleton(X,Y"), -skeleton(X,Y'), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

pushSkeletonLeftWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(X',X), skeleton(X',Y), plusOne(X",X'), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X',X), plusOne(X",X'), skeleton(X",Y), -skeleton(X',Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

pushSkeletonRightWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(X,X'), skeleton(X',Y), plusOne(X',X"), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X,X'), plusOne(X',X"), skeleton(X",Y), -skeleton(X',Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

wallKillUpWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(Y,Y'), skeleton(X,Y'), plusOne(Y',Y"), wall(X,Y")

Effect : plusOne(Y,Y'), -skeleton(X,Y'), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

rockKillUpWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(Y,Y'), skeleton(X,Y'), plusOne(Y',Y"), rock(X,Y")

Effect : plusOne(Y,Y'), -skeleton(X,Y'), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

wallKillDownWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(Y',Y), skeleton(X,Y'), plusOne(Y",Y'), wall(X,Y")

Effect : plusOne(Y',Y), -skeleton(X,Y'), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

rockKillDownWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(Y',Y), skeleton(X,Y'), plusOne(Y",Y'), rock(X,Y")

Effect : plusOne(Y',Y), -skeleton(X,Y'), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

wallKillLeftWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(X',X), skeleton(X',Y), plusOne(X",X'), wall(X",Y)

Effect : plusOne(X',X), -skeleton(X',Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

rockKillLeftWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(X',X), skeleton(X',Y), plusOne(X",X'), rock(X",Y)

Effect : plusOne(X',X), -skeleton(X',Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

wallKillRightWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(X,X'), skeleton(X',Y), plusOne(X',X"), wall(X",Y)

Effect : plusOne(X,X'), -skeleton(X',Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

rockKillRightWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), -openTrap(X,Y), -trap(X,Y,0), counter(C), -counter(-1), plusOne(X,X'), skeleton(X',Y), plusOne(X',X"), rock(X",Y)

Effect : plusOne(X,X'), -skeleton(X',Y), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

obtainKeyUpWithChangingTrapOpen(X,Y,A,B,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), lock(A,B,0), counter(C), -counter(-1), plusOne(Y,Y'), key(X,Y')

Effect : plusOne(Y,Y'), helltaker(X,Y'), -helltaker(X,Y), lock(A,B,1), -lock(A,B,0), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

obtainKeyDownWithChangingTrapOpen(X,Y,A,B,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), lock(A,B,0), counter(C), -counter(-1), plusOne(Y',Y), key(X,Y')

Effect : plusOne(Y',Y), helltaker(X,Y'), -helltaker(X,Y), lock(A,B,1), -lock(A,B,0), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

obtainKeyLeftWithChangingTrapOpen(X,Y,A,B,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), lock(A,B,0), counter(C), -counter(-1), plusOne(X',X), key(X',Y)

Effect : plusOne(X',X), helltaker(X',Y), -helltaker(X,Y), lock(A,B,1), -lock(A,B,0), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

obtainKeyRightWithChangingTrapOpen(X,Y,A,B,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), lock(A,B,0), counter(C), -counter(-1), plusOne(X,X'), key(X',Y)

Effect : plusOne(X,X'), helltaker(X',Y), -helltaker(X,Y), lock(A,B,1), -lock(A,B,0), plusOne(C',C), counter(C'), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

goUpOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y,Y'), -rock(X,Y'), -skeleton(X,Y'), openTrap(X,Y')

Effect : plusOne(Y,Y'), helltaker(X,Y'), -helltaker(X,Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

goDownOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y',Y), -rock(X,Y'), -skeleton(X,Y'), openTrap(X,Y')

Effect : plusOne(Y',Y), helltaker(X,Y'), -helltaker(X,Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

goLeftOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X',X), -rock(X',Y), -skeleton(X',Y), openTrap(X',Y)

Effect : plusOne(X',X), helltaker(X',Y), -helltaker(X,Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

goRightOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X,X'), -rock(X',Y), -skeleton(X',Y), openTrap(X',Y)

Effect : plusOne(X,X'), helltaker(X',Y), -helltaker(X,Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

pushRockUpWhileOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y,Y'), rock(X,Y'), plusOne(Y',Y"), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y,Y'), plusOne(Y',Y"), rock(X,Y"), -rock(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

pushRockDownWhileOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y',Y), rock(X,Y'), plusOne(Y",Y'), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y',Y), plusOne(Y",Y'), rock(X,Y"), -rock(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

pushRockLeftWhileOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X',X), rock(X',Y), plusOne(X",X'), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X',X), plusOne(X",X'), rock(X",Y), -rock(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

pushRockRightWhileOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X,X'), rock(X',Y), plusOne(X',X"), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X,X'), plusOne(X',X"), rock(X",Y), -rock(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

pushSkeletonUpWhileOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y,Y'), skeleton(X,Y'), plusOne(Y',Y"), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y,Y'), plusOne(Y',Y"), skeleton(X,Y"), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

pushSkeletonDownWhileOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y',Y), skeleton(X,Y'), plusOne(Y",Y'), -wall(X,Y"), -demon(X,Y"), -rock(X,Y"), -skeleton(X,Y"), -lock(X,Y",1), -lock(X,Y",0)

Effect : plusOne(Y',Y), plusOne(Y",Y'), skeleton(X,Y"), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

pushSkeletonLeftWhileOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X',X), skeleton(X',Y), plusOne(X",X'), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X',X), plusOne(X",X'), skeleton(X",Y), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

pushSkeletonRightWhileOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X,X'), skeleton(X',Y), plusOne(X',X"), -wall(X",Y), -demon(X",Y), -rock(X",Y), -skeleton(X",Y), -lock(X",Y,1), -lock(X",Y,0)

Effect : plusOne(X,X'), plusOne(X',X"), skeleton(X",Y), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

wallKillUpWhileOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y,Y'), skeleton(X,Y'), plusOne(Y',Y"), wall(X,Y")

Effect : plusOne(Y,Y'), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

rockKillUpWhileOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y,Y'), skeleton(X,Y'), plusOne(Y',Y"), rock(X,Y")

Effect : plusOne(Y,Y'), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

wallKillDownWhileOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y',Y), skeleton(X,Y'), plusOne(Y",Y'), wall(X,Y")

Effect : plusOne(Y',Y), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

rockKillDownWhileOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(Y',Y), skeleton(X,Y'), plusOne(Y",Y'), rock(X,Y")

Effect : plusOne(Y',Y), -skeleton(X,Y'), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

wallKillLeftWhileOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X',X), skeleton(X',Y), plusOne(X",X'), wall(X",Y)

Effect : plusOne(X',X), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

rockKillLeftWhileOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X',X), skeleton(X',Y), plusOne(X",X'), rock(X",Y)

Effect : plusOne(X',X), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

wallKillRightWhileOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X,X'), skeleton(X',Y), plusOne(X',X"), wall(X",Y)

Effect : plusOne(X,X'), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)

rockKillRightWhileOnOpenTrapWithChangingTrapOpen(X,Y,C,TX,TY,UX,UY)

Precondition : helltaker(X,Y),trap(TX,TY,1),trap(UX,UY,1), openTrap(X,Y), counter(C), -counter(-1), -counter(0), -counter(1), plusOne(X,X'), skeleton(X',Y), plusOne(X',X"), rock(X",Y)

Effect : plusOne(X,X'), -skeleton(X',Y), plusOne(C',C), plusOne(C",C'), counter(C"), -counter(C), -trap(TX,TY,1), -trap(UX,UY,1), trap(TX,TY,0), trap(UX,UY,0)