UNIVERSIDADE DA CORUÑA

Generación de Escenarios de un Videojuego 2D mediante Programación Lógica

Rafael Alcalde Azpiazu

Grado en Ingeniería Informática
Mención en Computación

Proyecto clásico de Ingeniería
Facultad de Informática

Director: Pedro Cabalar

A Coruña, 19 de septiembre de 2018

Motivación

- La industria del videojuego constituye un sector económico cada vez más relevante.

Motivación

- La industria del videojuego constituye un sector económico cada vez más relevante.

  Beneficios netos (en dólares):

  - Minecraft: 2500 millones, Fornite: 1000 millones, Destiny: 500 millones.

Motivación

- La industria del videojuego constituye un sector económico cada vez más relevante.

  Beneficios netos (en dólares):

  - Minecraft: 2500 millones, Fornite: 1000 millones, Destiny: 500 millones.

- Ha activado avances tecnológicos, p. ej. el uso de la Inteligencia Artificial, uno de los campos que más ha contribuido.

Motivación

- La industria del videojuego constituye un sector económico cada vez más relevante.

  Beneficios netos (en dólares):

  - Minecraft: 2500 millones, Fornite: 1000 millones, Destiny: 500 millones.

- Ha activado avances tecnológicos, p. ej. el uso de la Inteligencia Artificial, uno de los campos que más ha contribuido.

  - Diseño de enemigos inteligentes, p. ej mediante programación evolutiva (No Man's Sky).

Motivación

- La industria del videojuego constituye un sector económico cada vez más relevante.

  Beneficios netos (en dólares):

  - Minecraft: 2500 millones, Fornite: 1000 millones, Destiny: 500 millones.

- Ha activado avances tecnológicos, p. ej. el uso de la Inteligencia Artificial, uno de los campos que más ha contribuido.

  - Diseño de enemigos inteligentes, p. ej mediante programación evolutiva (No Man's Sky).
  - Diseño del entorno. Existen dos aproximaciones:

    - Generación procedimental. La más usada.

Motivación

- La industria del videojuego constituye un sector económico cada vez más relevante.

  Beneficios netos (en dólares):

  - Minecraft: 2500 millones, Fornite: 1000 millones, Destiny: 500 millones.

- Ha activado avances tecnológicos, p. ej. el uso de la Inteligencia Artificial, uno de los campos que más ha contribuido.

  - Diseño de enemigos inteligentes, p. ej mediante programación evolutiva (No Man's Sky).
  - Diseño del entorno. Existen dos aproximaciones:

    - Generación procedimental. La más usada.
    - Generación declarativa $\Leftarrow$.

Diseño de entornos

- Generación procedimental: se basa en un algoritmo o técnica ya predefinida.

  1. Algoritmo ad-hoc.
  2. Programación evolutiva.
  3. Expresiones matemáticas.

Diseño de entornos

- Generación procedimental: se basa en un algoritmo o técnica ya predefinida.
  1. Algoritmo ad-hoc.
  2. Programación evolutiva.
  3. Expresiones matemáticas.

Problemática

Para influir en el resultado de la generación se necesita reprogramar el algoritmo generador para adaptarlo a los criterios.

Diseño de entornos

- Generación declarativa: existe una representación formal del entorno, p. ej. mediante programación lógica.

Diseño de entornos

- Generación declarativa: existe una representación formal del entorno, p. ej. mediante programación lógica.
- La generación es independiente del algoritmo de búsqueda usado para obtener las posibles soluciones.

Diseño de entornos

- Generación declarativa: existe una representación formal del entorno, p. ej. mediante programación lógica.
- La generación es independiente del algoritmo de búsqueda usado para obtener las posibles soluciones.
- Un caso concreto de programación lógica es *Answer Set Programming*.

Diseño de entornos

- Generación declarativa: existe una representación formal del entorno, p. ej. mediante programación lógica.
- La generación es independiente del algoritmo de búsqueda usado para obtener las posibles soluciones.
- Un caso concreto de programación lógica es *Answer Set Programming*.
  - *Answer Set Programming for Procedural Content Generation: A Design Space Approach* [Smith et al, 11] (ASP + Warzone 2100).

Diseño de entornos

- Generación declarativa: existe una representación formal del entorno, p. ej. mediante programación lógica.
- La generación es independiente del algoritmo de búsqueda usado para obtener las posibles soluciones.
- Un caso concreto de programación lógica es *Answer Set Programming*.
  - *Answer Set Programming for Procedural Content Generation: A Design Space Approach* [Smith et al, 11] (ASP + Warzone 2100).
  - En este proyecto: ASP + Freeciv $\Leftarrow$.

Freeciv

- Versión *open source* y gratuita de Sid Meier's Civilization creado en la universidad de Aarhus.

Freeciv

- Versión *open source* y gratuita de Sid Meier's Civilization creado en la universidad de Aarhus.
- Juego de estrategia por turnos.

Freeciv

- Versión *open source* y gratuita de Sid Meier's Civilization creado en la universidad de Aarhus.
- Juego de estrategia por turnos.
- El jugador controla a un grupo de colonos, comienza en el año 4000 A.C.

Freeciv

- Versión *open source* y gratuita de Sid Meier's Civilization creado en la universidad de Aarhus.
- Juego de estrategia por turnos.
- El jugador controla a un grupo de colonos, comienza en el año 4000 A.C.
- El objetivo final es crear una gran civilización. Para ello existen 5 formas de finalizar el juego:
  - Victoria por dominación, científica, religión, cultural o por puntuación.

Tipos de terrenos en Freeciv

- Hay 12 tipos de terreno, con posibles bonificaciones.

Objetivos del proyecto

Para este proyecto:

- Se definirá un modelo declarativo del escenario para Freeciv usando *Answer Set Programming*.

Objetivos del proyecto

Para este proyecto:

- Se definirá un modelo declarativo del escenario para Freeciv usando *Answer Set Programming*.
- Se construirá una pequeña herramienta gráfica con la que manipular el escenario.

Objetivos del proyecto

Para este proyecto:

- Se definirá un modelo declarativo del escenario para Freeciv usando *Answer Set Programming*.
- Se construirá una pequeña herramienta gráfica con la que manipular el escenario.
- Eficiencia: reducir o podar el número de combinaciones posibles.

Índice

Índice

Answer Set Programming

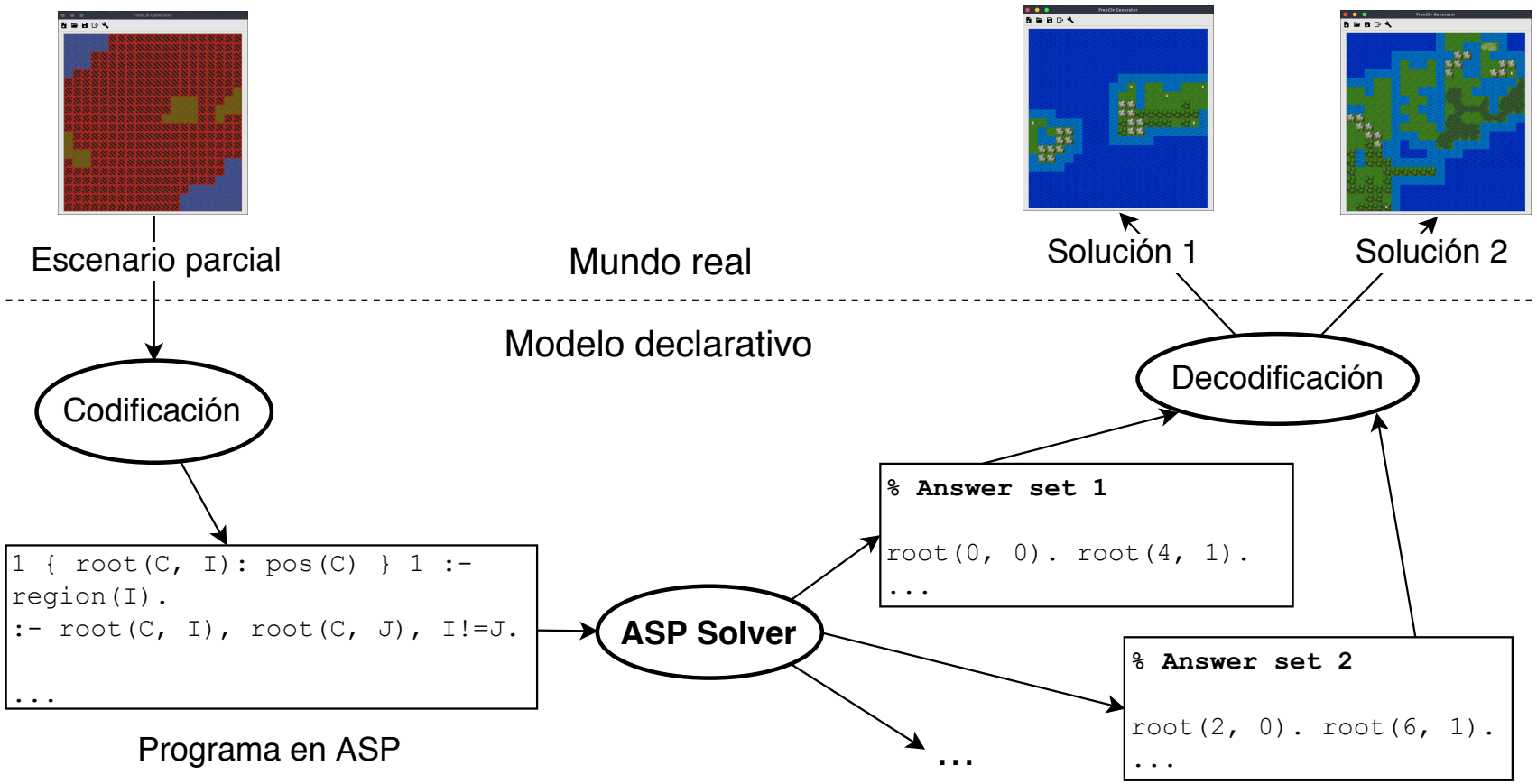
- Paradigma enfocado a la resolución declarativa de problemas con complejidad *NP-hard*.

Answer Set Programming

- Paradigma enfocado a la resolución declarativa de problemas con complejidad *NP-hard*.
- Combina un lenguaje simple (predicados y reglas) con el que modelar problemas lógicos y herramientas de alto rendimiento.

Answer Set Programming

- Paradigma enfocado a la resolución declarativa de problemas con complejidad *NP-hard*.
- Combina un lenguaje simple (predicados y reglas) con el que modelar problemas lógicos y herramientas de alto rendimiento.



Generación del escenario

- Generar todo el terreno a la vez es inviable.

Generación del escenario

- Generar todo el terreno a la vez es inviable.
- Se divide el mapa en cuadrantes y regiones.
  - Un cuadrante es un grupo de celdas

Generación del escenario

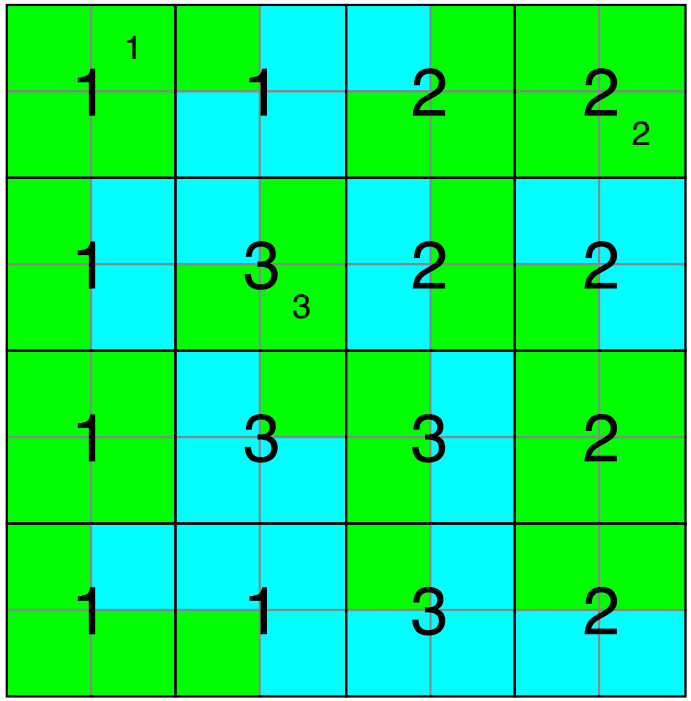
- Generar todo el terreno a la vez es inviable.
- Se divide el mapa en cuadrantes y regiones.
  - Un cuadrante es un grupo de celdas
  1. Se genera regiones, que son grupos de cuadrantes conectados entre sí.

1	1	2	2
1	3	2	2
1	3	3	2
1	1	3	2

Generación del escenario

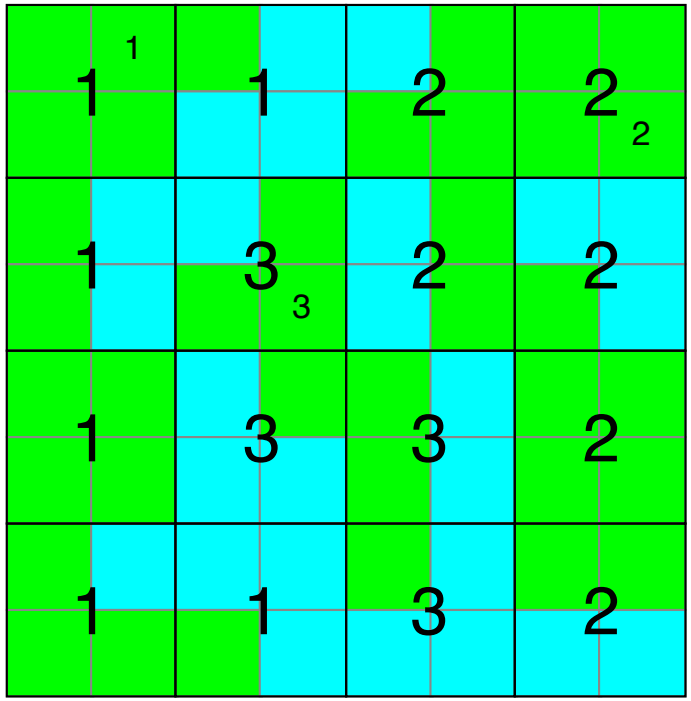
- Generar todo el terreno a la vez es inviable.
- Se divide el mapa en cuadrantes y regiones.
  - Un cuadrante es un grupo de celdas

1. Se genera regiones, que son grupos de cuadrantes conectados entre sí.
2. Se detalla el terreno de cada región, formando islas.



Generación del escenario

- Generar todo el terreno a la vez es inviable.
- Se divide el mapa en cuadrantes y regiones.
  - Un cuadrante es un grupo de celdas
  1. Se genera regiones, que son grupos de cuadrantes conectados entre sí.
  2. Se detalla el terreno de cada región, formando islas.
- Los módulos son independientes. No necesitan toda la información del problema.



Ejemplo de algunos predicados usados

`root(C, I)`	Cuadrante inicial `C` de la región `I`.
`pos(C)`	Posición `C` de un cuadrante.
`reached(C, I)`	Cuadrante `C` que ha sido alcanzado.
`adj(C, D)`	Posiciones adyacentes `C` y `D` en la cuadrícula.
`land(C)`	Celda `C` definida como tierra.
`mountain(C)`	Celda `C` definida como montaña.

Ejemplo de algunas reglas lógicas

Generación del cuadrantes

```
reached(C, I) :- root(C, I).
```

Ejemplo de algunas reglas lógicas

Generación del cuadrantes

```
reached(C, I) :- root(C, I).

0 { reached(C, I) } 1 :- region(I), reached(D, I),
    adj(D, C), not existsanother(C, I).
```

Ejemplo de algunas reglas lógicas

Generación del cuadrantes

```
reached(C, I) :- root(C, I).

0 { reached(C, I) } 1 :- region(I), reached(D, I),
    adj(D, C), not existsanother(C, I).

1 { root(C, I): pos(C) } 1 :- region(I).
```

Ejemplo de algunas reglas lógicas

Generación del cuadrantes

```
reached(C, I) :- root(C, I).

0 { reached(C, I) } 1 :- region(I), reached(D, I),
    adj(D, C), not existsanother(C, I).

1 { root(C, I): pos(C) } 1 :- region(I).

:- root(C, I), root(C, J), region(I), region(J), I!=J.
```

Ejemplo de algunas reglas lógicas

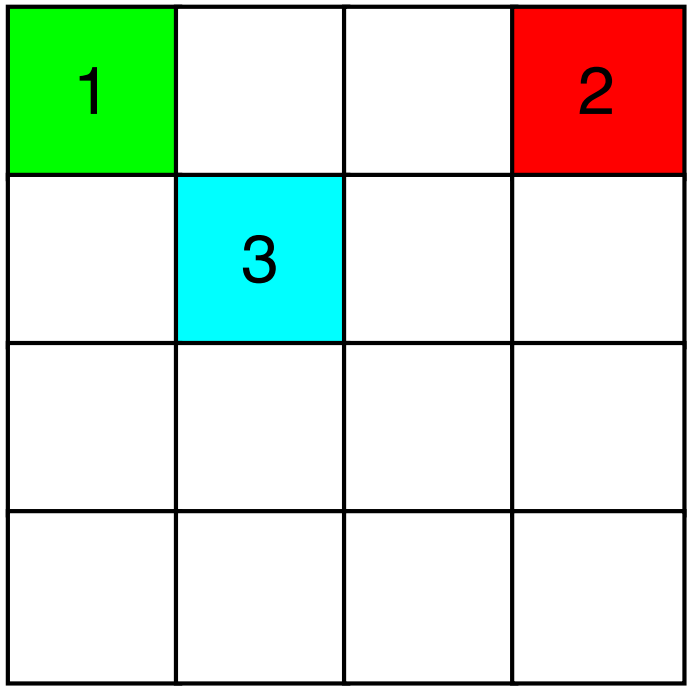
Generación del cuadrantes

```
reached(C, I) :- root(C, I).

0 { reached(C, I) } 1 :-
    region(I), reached(D, I),
    adj(D, C),
    not existsanother(C, I).

1 { root(C, I): pos(C) } 1 :-
    region(I).

:- root(C, I), root(C, J),
    region(I), region(J), I!=J.
```



Ejemplo de algunas reglas lógicas

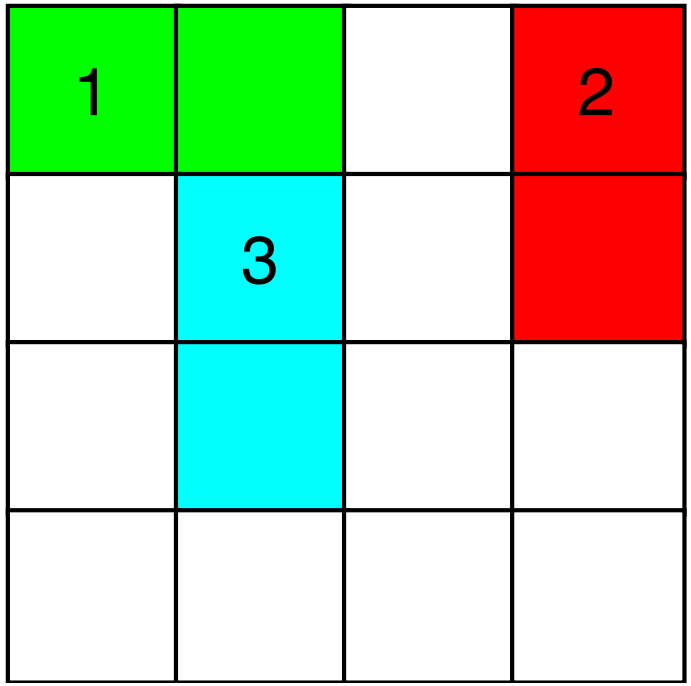
Generación del cuadrantes

```
reached(C, I) :- root(C, I).

0 { reached(C, I) } 1 :-
    region(I), reached(D, I),
    adj(D, C),
    not existsanother(C, I).

1 { root(C, I): pos(C) } 1 :-
    region(I).

:- root(C, I), root(C, J),
    region(I), region(J), I!=J.
```



Ejemplo de algunas reglas lógicas

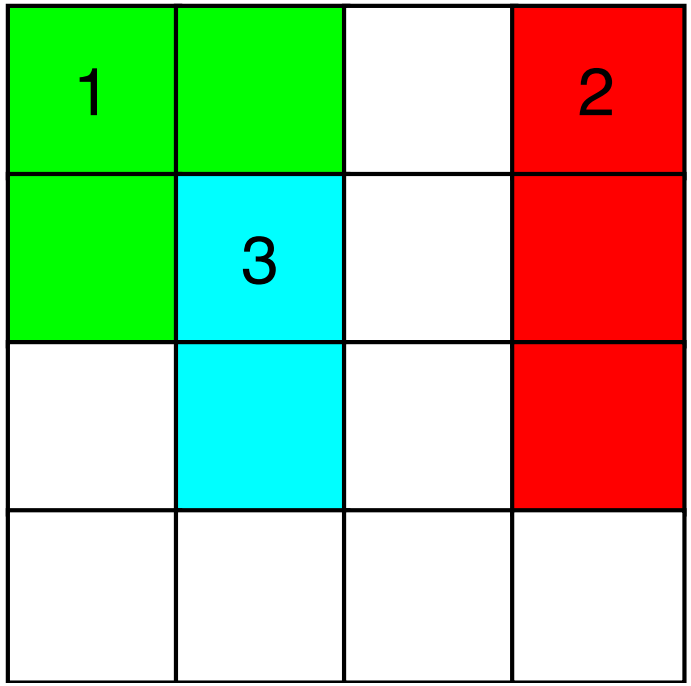
Generación del cuadrantes

```
reached(C, I) :- root(C, I).

0 { reached(C, I) } 1 :-
    region(I), reached(D, I),
    adj(D, C),
    not existsanother(C, I).

1 { root(C, I): pos(C) } 1 :-
    region(I).

:- root(C, I), root(C, J),
    region(I), region(J), I!=J.
```



Ejemplo de algunas reglas lógicas

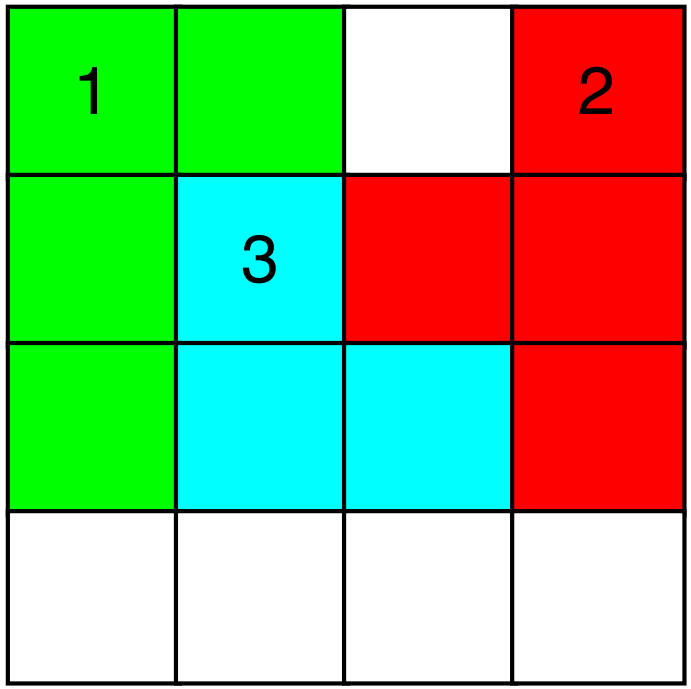
Generación del cuadrantes

```
reached(C, I) :- root(C, I).

0 { reached(C, I) } 1 :-
    region(I), reached(D, I),
    adj(D, C),
    not existsanother(C, I).

1 { root(C, I): pos(C) } 1 :-
    region(I).

:- root(C, I), root(C, J),
    region(I), region(J), I!=J.
```



Ejemplo de algunas reglas lógicas

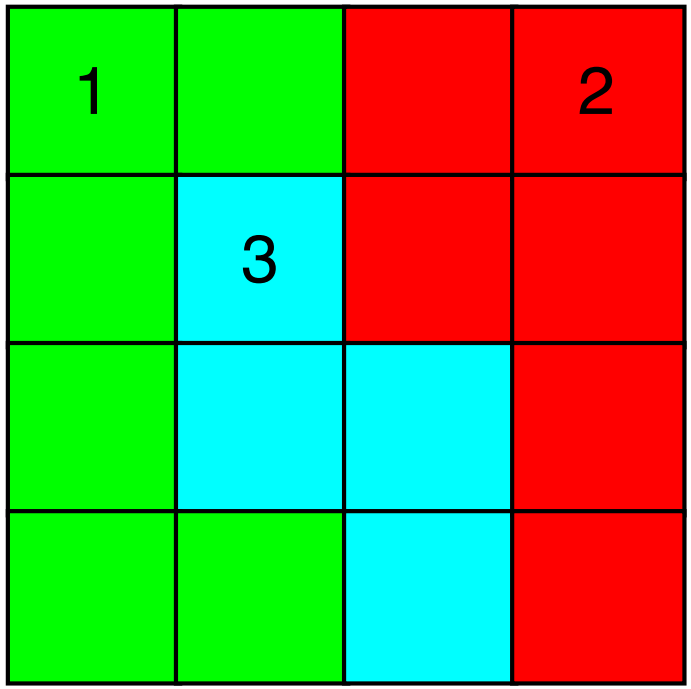
Generación del cuadrantes

```
reached(C, I) :- root(C, I).

0 { reached(C, I) } 1 :-
    region(I), reached(D, I),
    adj(D, C),
    not existsanother(C, I).

1 { root(C, I): pos(C) } 1 :-
    region(I).

:- root(C, I), root(C, J),
    region(I), region(J), I!=J.
```



Generación de biomas y de puntos de inicio

- Generación de biomas:
  - Un bioma es una zona con el mismo tipo de terreno.

Generación de biomas y de puntos de inicio

- Generación de biomas:
  - Un bioma es una zona con el mismo tipo de terreno.
  - Misma forma que la generación de islas, salvo que se pueden pegar.

Generación de biomas y de puntos de inicio

- Generación de biomas:
  - Un bioma es una zona con el mismo tipo de terreno.
  - Misma forma que la generación de islas, salvo que se pueden pegar.

- Generación de puntos de inicio:
  - Se escoge N celdas de tierra para los jugadores.

Generación de biomas y de puntos de inicio

- Generación de biomas:
  - Un bioma es una zona con el mismo tipo de terreno.
  - Misma forma que la generación de islas, salvo que se pueden pegar.

- Generación de puntos de inicio:
  - Se escoge N celdas de tierra para los jugadores.
  - Para estos puntos se añaden preferencias:
    - Minimizar la distancia al agua.

Generación de biomas y de puntos de inicio

- Generación de biomas:
  - Un bioma es una zona con el mismo tipo de terreno.
  - Misma forma que la generación de islas, salvo que se pueden pegar.

- Generación de puntos de inicio:
  - Se escoge N celdas de tierra para los jugadores.
  - Para estos puntos se añaden preferencias:
    - Minimizar la distancia al agua.
    - Maximizar la distancia a las montañas.

Índice

Índice

Arquitectura del sistema

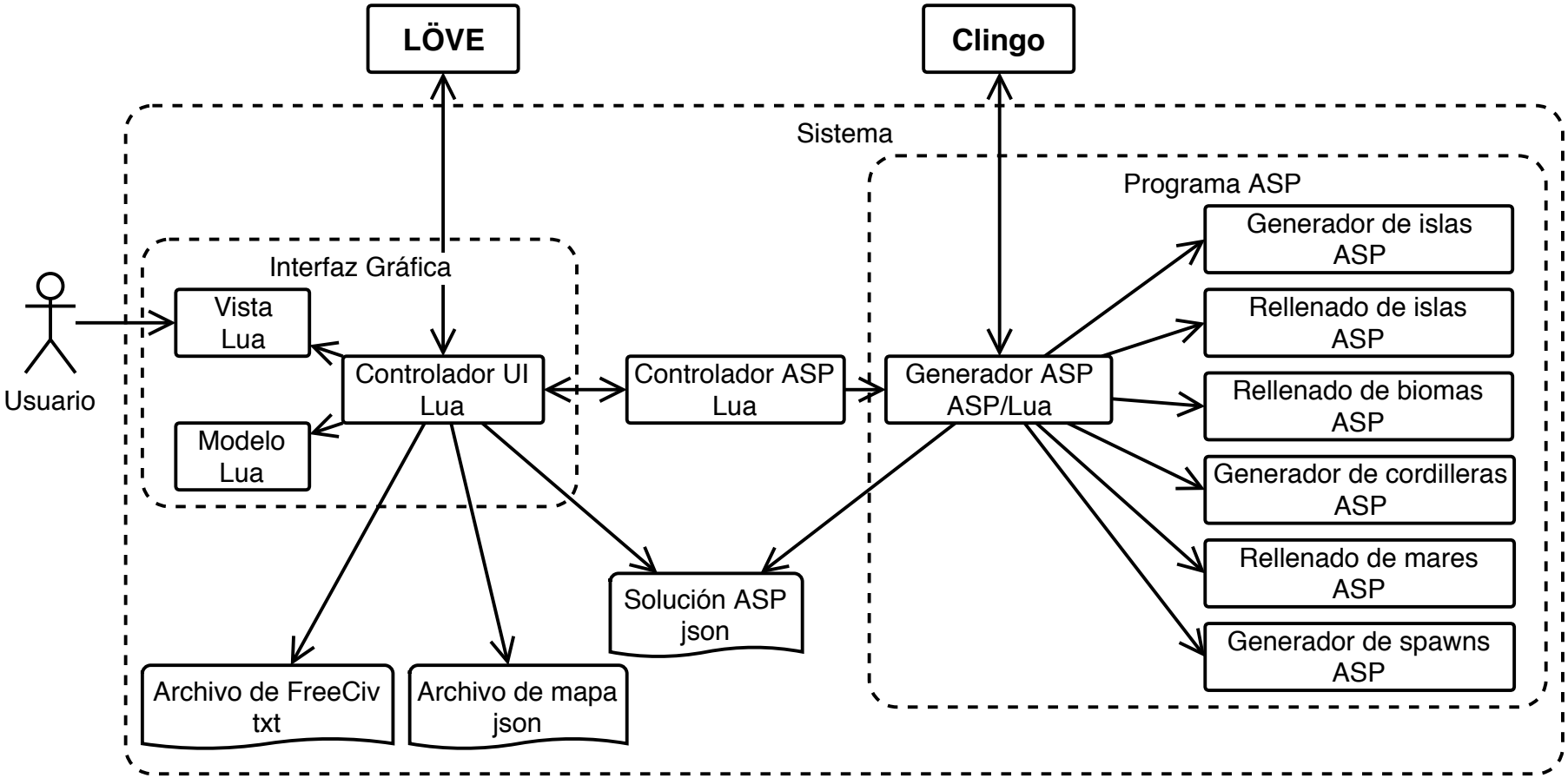
- *Front-end*: Creado en LÖVE, usa un modelo MVC que usa una interfaz gráfica en modo inmediato.

Arquitectura del sistema

- *Front-end*: Creado en LÖVE, usa un modelo MVC que usa una interfaz gráfica en modo inmediato.
- *Back-end*: Creado en Lua (con API de clingo) y ASP, usa un modelo en *pipeline*.

Arquitectura del sistema

- *Front-end*: Creado en LÖVE, usa un modelo MVC que usa una interfaz gráfica en modo inmediato.
- *Back-end*: Creado en Lua (con API de clingo) y ASP, usa un modelo en *pipeline*.



Proceso de ingeniería

- Metodología: desarrollo iterativo incremental y evolutivo.

Proceso de ingeniería

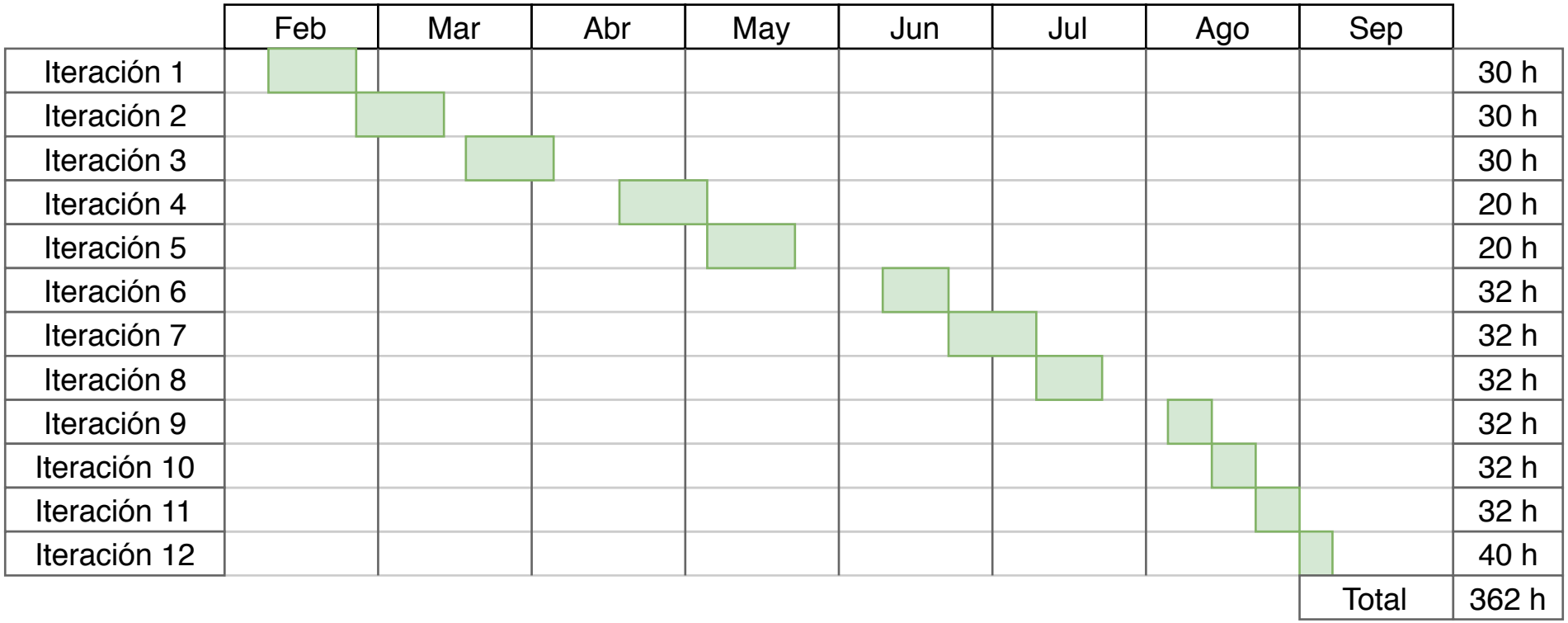
- Metodología: desarrollo iterativo incremental y evolutivo.
- Se ha usado herramientas de uso libre y gratuitas para el desarrollo del proyecto.

Proceso de ingeniería

- Metodología: desarrollo iterativo incremental y evolutivo.
- Se ha usado herramientas de uso libre y gratuitas para el desarrollo del proyecto.
- El coste total del proyecto asciende a 3720.00 €.

Proceso de ingeniería

- Metodología: desarrollo iterativo incremental y evolutivo.
- Se ha usado herramientas de uso libre y gratuitas para el desarrollo del proyecto.
- El coste total del proyecto asciende a 3720.00 €.



	Feb	Mar	Abr	May	Jun	Jul	Ago	Sep	
Iteración 1									30 h
Iteración 2									30 h
Iteración 3									30 h
Iteración 4									20 h
Iteración 5									20 h
Iteración 6									32 h
Iteración 7									32 h
Iteración 8									32 h
Iteración 9									32 h
Iteración 10									32 h
Iteración 11									32 h
Iteración 12									40 h
								Total	362 h

Índice

Evaluación

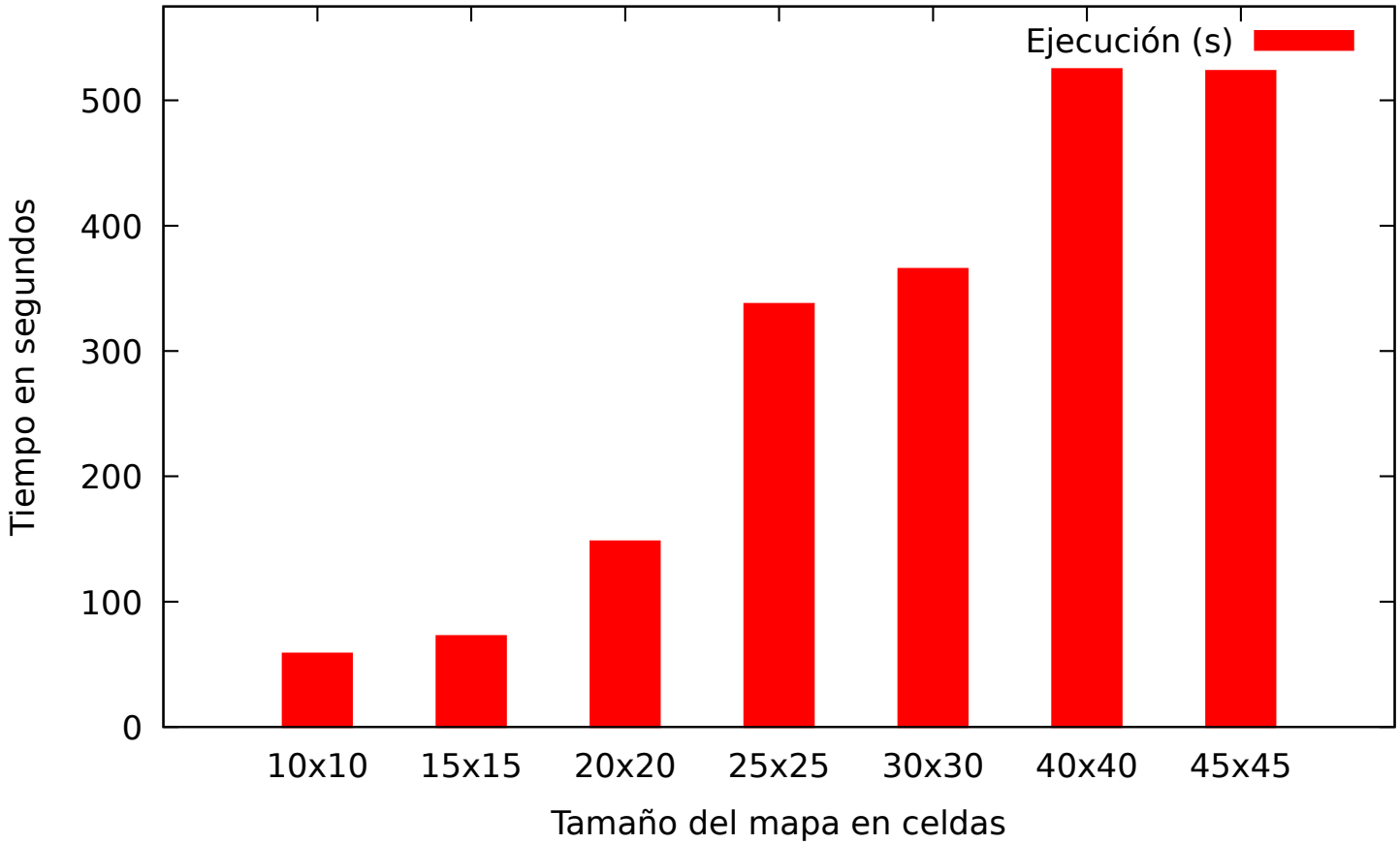
- Se han procedido a realizar diferentes pruebas con distintas variables.

Evaluación

- Se han procedido a realizar diferentes pruebas con distintas variables.
- Por ejemplo, definiendo distintos tamaños de escenarios.

Evaluación

- Se han procedido a realizar diferentes pruebas con distintas variables.
- Por ejemplo, definiendo distintos tamaños de escenarios.



Evaluación

- Modificado porcentajes de terreno con tierra y tamaño de biomas.

Evaluación

- Modificado porcentajes de terreno con tierra y tamaño de biomas.
- Se observa la misma tendencia que en la primera prueba.

Tamaño de tierra	Tamaño de biomas: 10 %	15 %	20 %	25 %	30 %	35 %
10 %	42 s	41 s	35 s	88 s	117 s	214 s
15 %	41 s	37 s	36 s	88 s	133 s	247 s
20 %	37 s	43 s	37 s	105 s	123 s	224 s
25 %	29 s	51 s	43 s	92 s	123 s	220 s
30 %	42 s	43 s	138 s	95 s	-	-
35 %	41 s	40 s	38 s	90 s	-	241 s
40 %	43 s	46 s	36 s	-	-	217 s
45 %	69 s	49 s	36 s	737 s	-	220 s
50 %	100 s	-	1064 s	-	-	-
55 %	55 s	956 s	-	-	-	-
60 %	164 s	740 s	-	-	-	-
65 %	97 s	838 s	-	-	-	-

Índice

Conclusiones

- El sistema permite definir nuevas propiedades de forma sencilla.
  - Esto proporciona flexibilidad a la hora de adaptar el sistema.
  - No hay que tener en cuenta el método de resolución.
- Se ha reducido el problema de eficiencia ocasionado por el número exponencial de combinaciones posibles.

Trabajo futuro

- Añadir otros elementos del mapa al generador (ríos, comida, animales, etc...).
- Mejorar el rendimiento del sistema.
- Posible mejora en la interfaz gráfica.
  - Mejorar la manipulación del mapa.
  - Añadir nuevos tipos de restricciones.
- Publicar la herramienta para tener una base de usuarios.

UNIVERSIDADE DA CORUÑA

Generación de Escenarios de un Videojuego 2D mediante Programación Lógica

Rafael Alcalde Azpiazu

¡Gracias por su atención!

Grado en Ingeniería Informática
Mención en Computación

Proyecto clásico de Ingeniería
Facultad de Informática

Director: Pedro Cabalar

Ejemplo de la biblioteca de Clingo

Añadir restricciones al programa

```
#script(lua)
function main(prog)
    --Genero las regiones
    for i = 0, c_regions-1 do
        --Hace grounding del programa lógico
        prog:ground({{"base", {}}, {"generate", {i}}})
        --Obtengo un manejador de la solución
        handle = prog:solve(yield=true)

        local restrictions = " "
        --Recorre los modelos de la solución
        for model in handle:iter() do
            --Añado las restricciones
            if #restrictions ~= 0 then
                prog1:load("resources/restrictions.lp")
            end
            ...
```

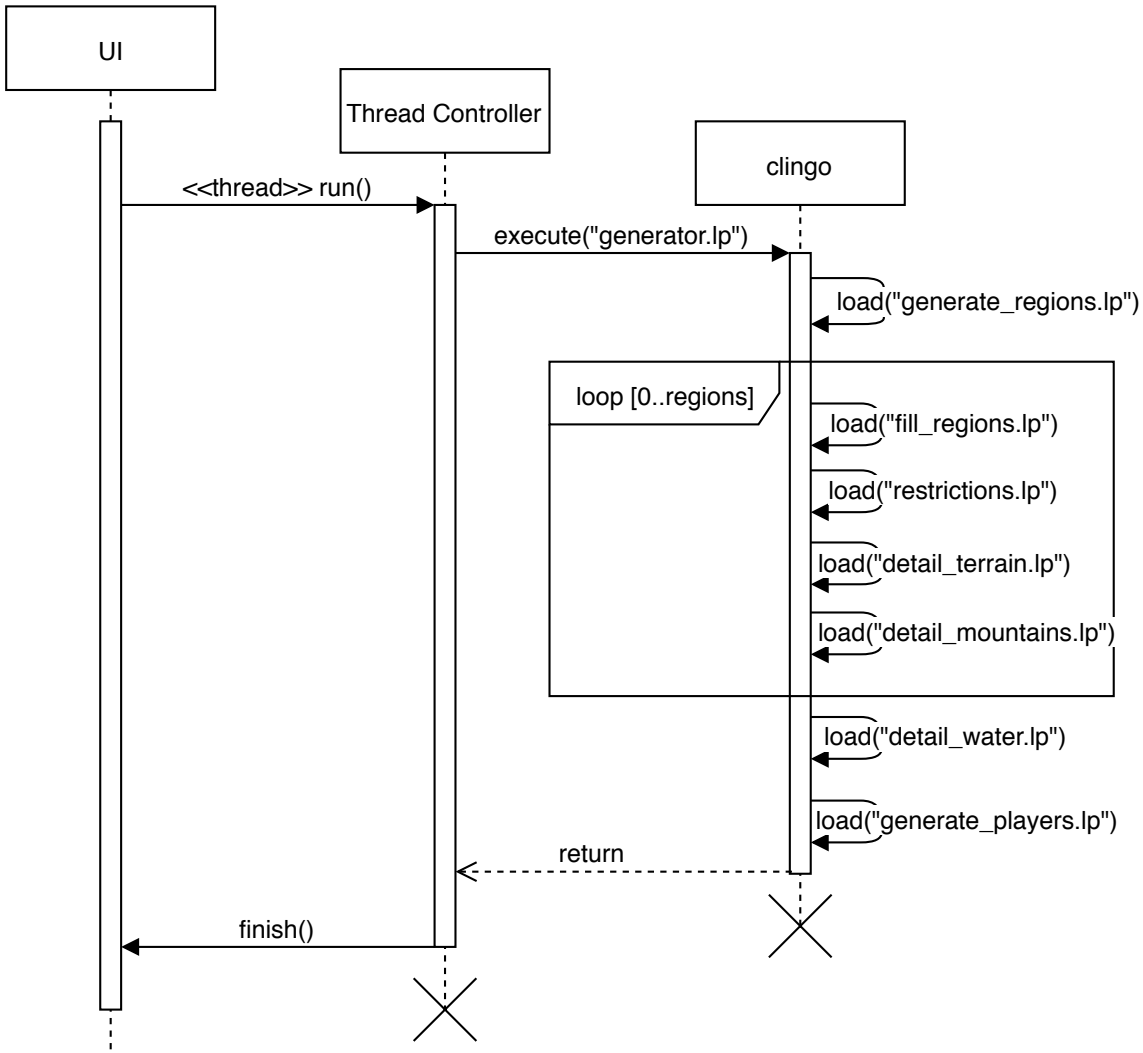
Ejemplo de la biblioteca de Clingo

Añadir restricciones al programa

```
            for m in handle:iter() do
               for row_str, col_str, contain in
string.gmatch(tostring(model), "cell%(p%((%d+),(%d+)%),(%l+)%)") do
                  if contain == "l"then
                     lands = lands .. "
land(p("..row_str..","..col_str..")).”
                     restrictions = restrictions ..
check_restrictions(row, col, i)
                  end
               end

               df = io.open("resources/restrictions.lp", "w+")
               df:write(restrictions)
               df:flush()
               df:close()
            end
         end
      end
end
#end.
```

Ejecución del módulo Clingo



Ejemplo de generación 20x20

Ejemplo de generación 30x30