

# Narratives and Interactive Storytelling

## Lecture 02 – Writing Interactive Narratives

Edirlei Soares de Lima

<edirlei.lima@universidadeeuropeia.pt>



# Authoring Interactive Narratives

- Interactive narratives can be created manually by an author or automatically generated by algorithms and simulations.
- **Branching narratives:**
  - A hand-crafted structure of nodes, often in the form of a graph/network, defines the possible storylines.
  - Each node includes a finely-crafted description of the plot event and the connections between nodes represent the possible paths that the story can follow.
  - The author's vision is precisely preserved.
  - Problem: manually authoring complex narratives is a complex task.

# Authoring Interactive Narratives

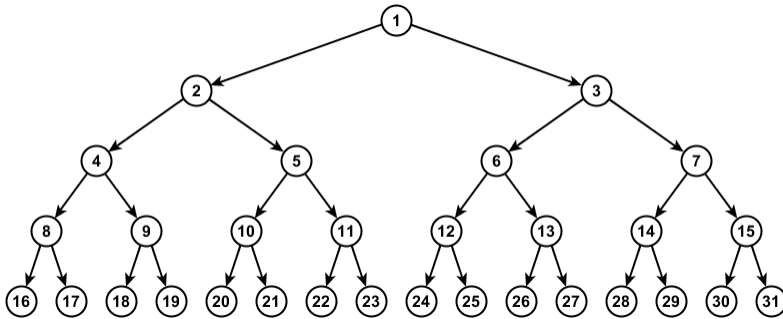
- Interactive narratives can be created manually by an author or automatically generated by algorithms and simulations.
- **Narratives generated by algorithms:**
  - A more robust forms of interactive storytelling.
  - Combines narrative theories with computer algorithms.
  - Two approaches: plot-based and character-based.
  - When authoring this form of narrative, the author defines only the story characters, a set of narrative events (with preconditions and effects), and an initial state of the world.
  - A planning algorithm is responsible for finding coherent sequences of events that will form the narrative.

# Writing Branching Narratives

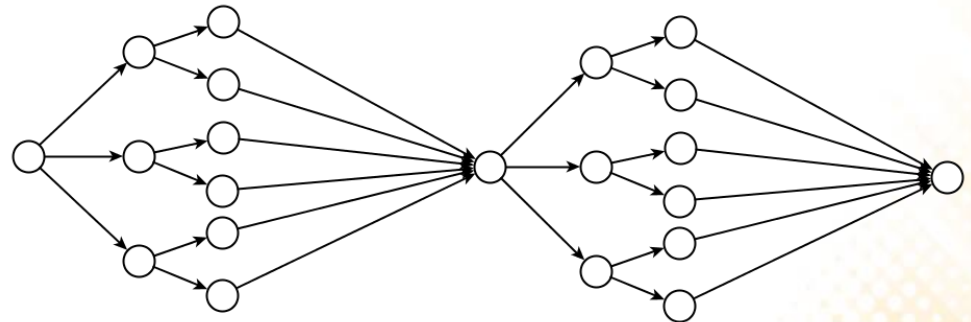
- **Elements:**

- Node: includes a description of the plot event.
- Connection: represent a possible path that the story can follow.

- **Structures:**



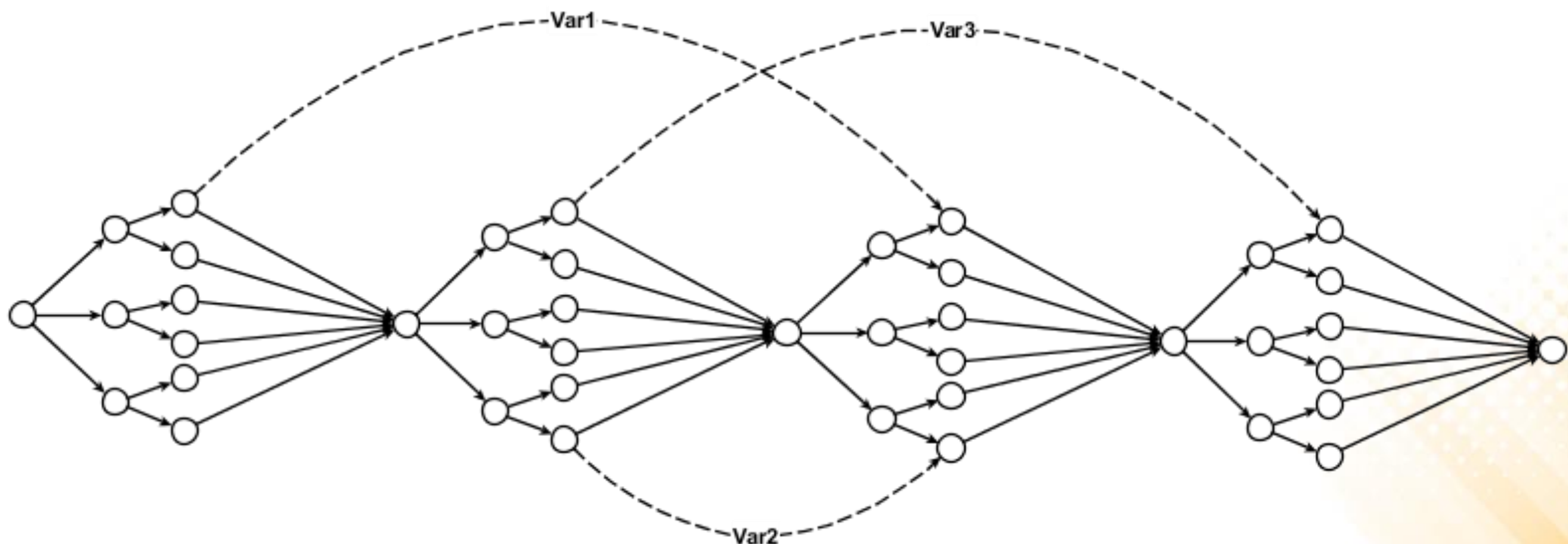
Branching Tree



Branching Network

# Writing Branching Narratives

- In a branching network, all branches can eventually lead to the same node (reduces complexity). But what is the point of choosing one path over another, if they both eventually lead to the same conclusion?
  - Solution: world state tracking.
  - Examples of variables to track: character status, personality, skills, morality, resources, ...

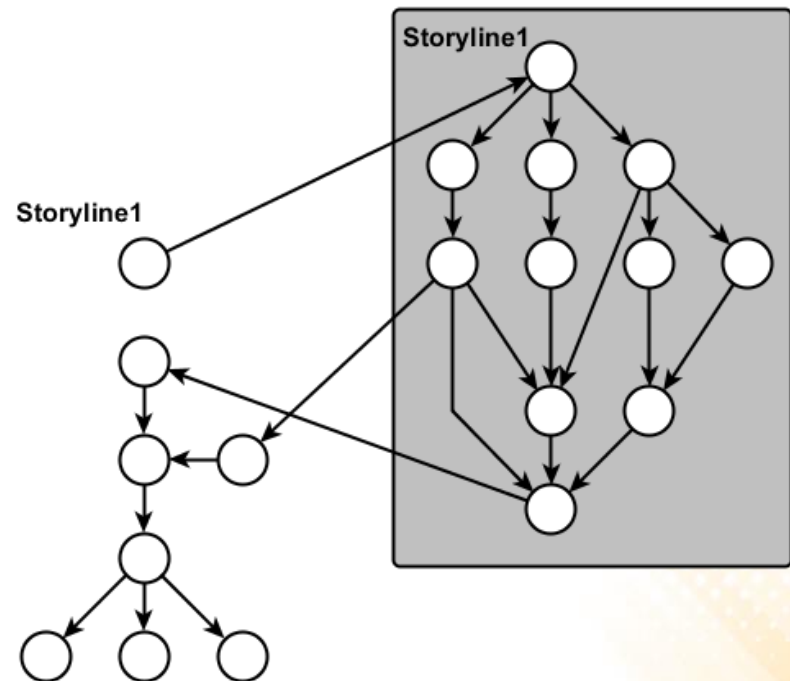
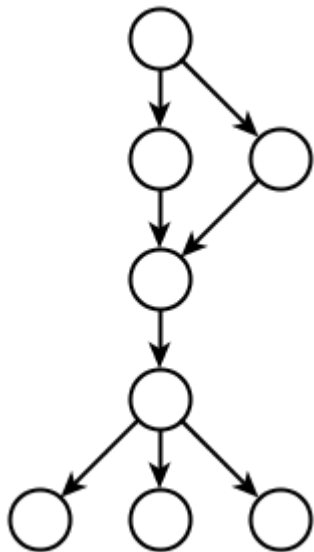


# Writing Branching Narratives

- **How to design the user choices (interaction points)?** The choices presented to users will largely define how they perceive the story.
- Some advices:
  - Choices need to have some effect on the story (otherwise there is no point for them).
  - Choices must start a new branch of the story or change some world state variable that affects the narrative latter.
  - The user must have some idea of how the choices will affect the story.
    - If he/she selects an option and something completely random happens and changes the story, the user will get frustrated.

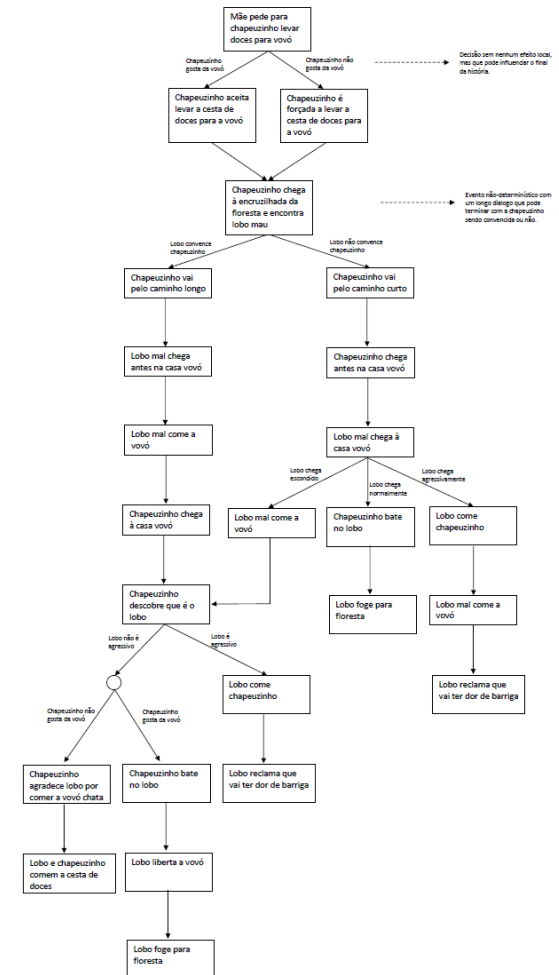
# Writing Branching Narratives

- Writing/designing recommendations:
  - Start with a high-level linear storyline;
  - Next, add branches to represent the possible ends of the story;
  - Zoom into each node of the high-level network, and start adding some branches within that node.



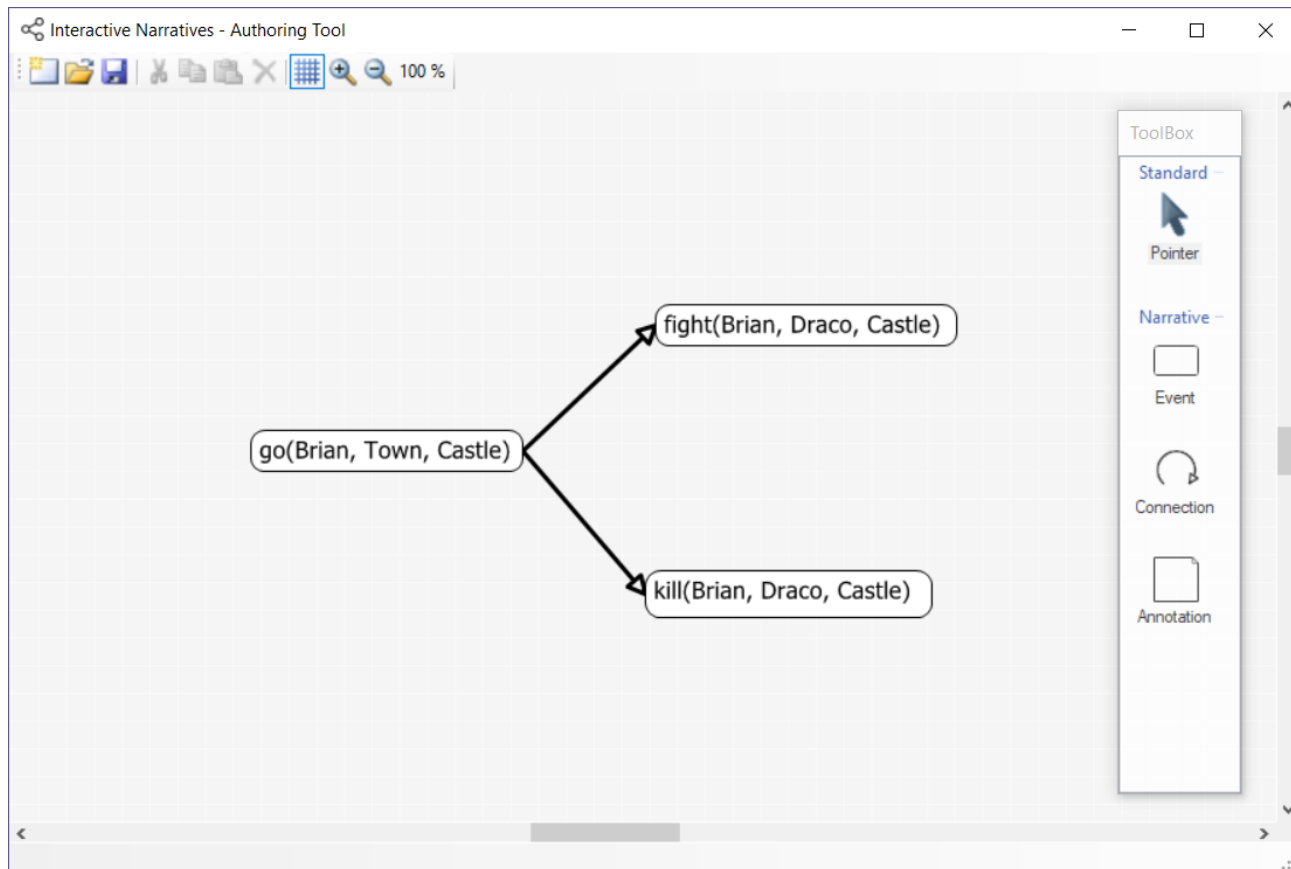
# Writing Branching Narratives

- **Example: Modern Little Red Riding Hood**





# Interactive Narrative Design Tool



<http://www.inf.puc-rio.br/~elima/is/INDesign.zip>

Online alternative: <https://www.draw.io/>

# Exercise 1

1) Write a branching storyline for the following idea:

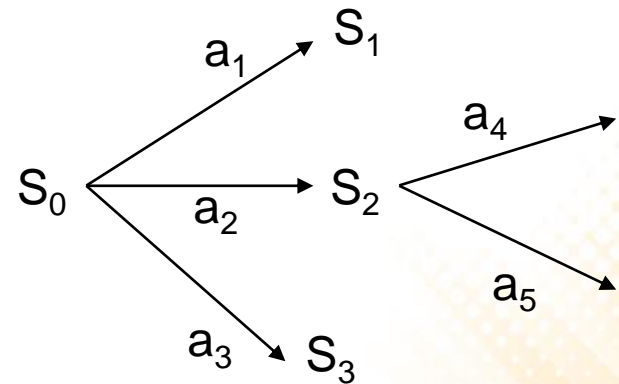
*“Once upon a time there was a charming princess, called Marian, lady of the White Palace, and two brave young men, sir Brian and sir Hoel, knights of the Gray Castle. Not far away in the sinister Red Castle, lives Draco, the evil dragon, ready to seize the princess, despite her guardians, and keep her with super-human strength. But there was also the silent wizard of the Green Forest, Turjan the mage. Whoever approached him with due courtesy could hope for a gift of great fighting power. Uncountable stories can be told in this world of fantasy. Will the princess be abducted by the dragon? Or killed by the monster? Will one of the knights save her or revenge her death, with or without the mage's help?”*

# Using Algorithms to Generate Interactive Narratives

- The process of generating stories involves narrative theories and algorithms.
  - The narrative theory gives the formalism on how the story is structured and the algorithms are responsible for generating coherent sequences of events to compose the plot.
- Narrative theories: Propp, Barthes, hero's journey, ...
- Algorithms: Hierarchical Task Networks (HTN), Heuristic Search Planners (HSP), first order logic planners, ...

# Automated Planning

- Planning is the task of finding a sequence of actions (a plan) to achieve a goal.
- **Planning problem elements:**
  - Initial State;
  - Actions (with preconditions and effects);
  - Goal;



# Automated Planning

- **Example:**

- Initial State:  $\text{character}(\text{brian}) \wedge \text{place}(\text{castle}) \wedge \text{place}(\text{store}) \wedge \text{item}(\text{sword}) \wedge \text{at}(\text{brian}, \text{castle}) \wedge \text{sell}(\text{store}, \text{sword})$
- Goal:  $\text{has}(\text{brian}, \text{sword}) \wedge \text{at}(\text{brian}, \text{castle})$
- Operator 1:  $\text{go}(\text{CH}, \text{PL1}, \text{PL2})$ 
  - precond:  $\text{character}(\text{CH}) \wedge \text{place}(\text{PL1}), \text{place}(\text{PL2}) \wedge \text{at}(\text{CH}, \text{PL1})$
  - effects:  $\neg \text{at}(\text{CH}, \text{PL1}) \wedge \text{at}(\text{CH}, \text{PL2})$
- Operator 2:  $\text{buy}(\text{CH}, \text{IT}, \text{PL})$ 
  - precond:  $\text{character}(\text{CH}) \wedge \text{item}(\text{IT}), \text{place}(\text{PL}) \wedge \text{at}(\text{CH}, \text{PL}) \wedge \text{sell}(\text{PL}, \text{IT})$
  - effects:  $\text{has}(\text{CH}, \text{IT})$
- Plan:  $\text{go}(\text{brian}, \text{castle}, \text{store}), \text{buy}(\text{brian}, \text{sword}, \text{store}), \text{go}(\text{brian}, \text{store}, \text{castle}).$

# Planning Domain Definition Language

- A planning problem is usually represented through a planning language, such as the PDDL (Planning Domain Definition Language).
  - PDDL was derived from the original STRIPS model, which is slightly more restrictive.
- Planning problems specified in PDDL are defined in two files:
  - Domain File: types, predicates, and actions.
  - Problem File: objects, initial state, and goal.

# PDDL – Syntax

- **Domain File:**

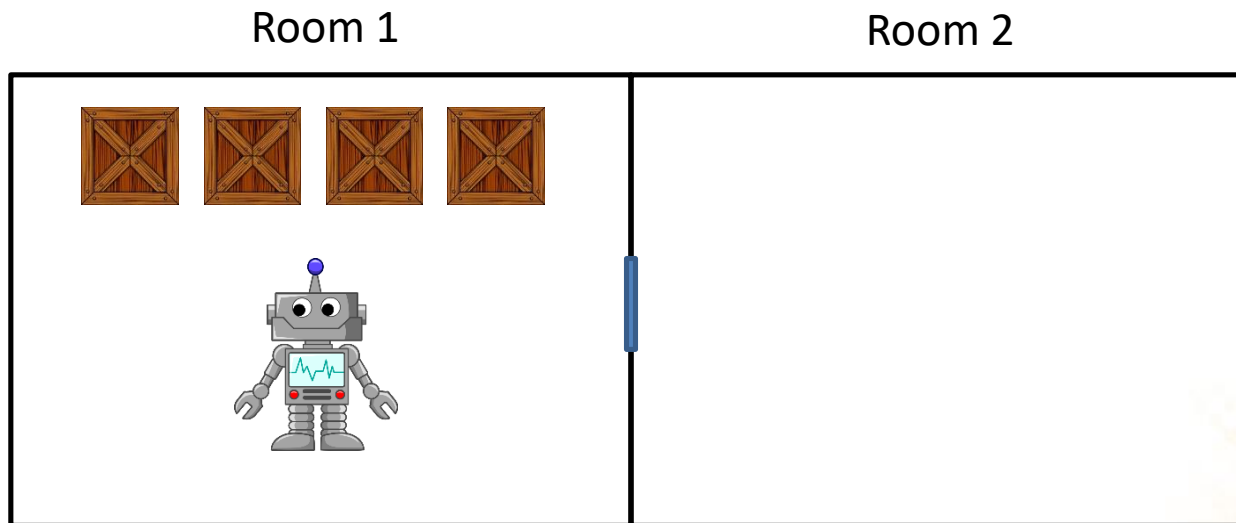
```
(define (domain <domain name>)
  (:requirements :strips :equality :typing)
  (:types <list of types>)
  (:constants <list of constants>)
  <PDDL code for predicates>
  <PDDL code for first action>
  [...]
  <PDDL code for last action>
)
```

- **Problem File:**

```
(define (problem <problem name>)
  (:domain <domain name>)
  <PDDL code for objects>
  <PDDL code for initial state>
  <PDDL code for goal specification>
)
```

# PDDL – Example Problem

- “There is robot that can move between two rooms and pickup/putdown boxes with two arms. Initially, the robot and 4 boxes are at room 1. The robot must take all boxes to room 2.”





# PDDL – Domain File

- **Types:**

```
(:types room box arm)
```

- **Constants:**

```
(:constants left right - arm)
```

- **Predicates:**

- robot-at(x) – true if the robot is at room x;
- box-at(x, y) – true if the box x is at room y;
- free(x) – true if the arm x is not holding a box;
- carry(x, y) – true if the arm x is holding a box y;

```
(:predicates  
  (robot-at ?x - room)  
  (box-at ?x - box ?y - room)  
  (free ?x - arm)  
  (carry ?x - box ?y - arm)  
)
```

# PDDL – Domain File

- Action: move the robot from room x to room y.
- Precondition: robot-at(x) must be true.
- Effect: robot-at(y) becomes true and robot-at(x) becomes false.

```
(:action move
  :parameters (?x ?y - room)
  :precondition (robot-at ?x)
  :effect (and (robot-at ?y) (not (robot-at ?x))))
)
```

# PDDL – Domain File

- Pickup Action:

```
(:action pickup
  :parameters (?x - box ?y - arm ?w - room)
  :precondition (and (free ?y) (robot-at ?w)
                    (box-at ?x ?w))
  :effect (and (carry ?x ?y) (not (box-at ?x ?w))
              (not(free ?y)))
)
```

- Putdown Action:

```
(:action putdown
  :parameters (?x - box ?y -arm ?w - room)
  :precondition (and (carry ?x ?y) (robot-at ?w))
  :effect (and (not(carry ?x ?y)) (box-at ?x ?w)
              (free ?y))
)
```

# PDDL – Domain File

```
(define (domain robot)
  (:requirements :strips :equality :typing)
  (:types room box arm)
  (:constants left right - arm)
  (:predicates
    (robot-at ?x - room)
    (box-at ?x - box ?y - room)
    (free ?x - arm)
    (carry ?x - box ?y - arm)
  )

  (:action move
    :parameters (?x ?y - room)
    :precondition (robot-at ?x)
    :effect (and (robot-at ?y) (not (robot-at ?x)))
  )

  (:action pickup
    :parameters (?x - box ?y - arm ?w - room)
    :precondition (and (free ?y) (robot-at ?w) (box-at ?x ?w))
    :effect (and (carry ?x ?y) (not (box-at ?x ?w)) (not(free ?y)))
  )

  (:action putdown
    :parameters (?x - box ?y -arm ?w - room)
    :precondition (and (carry ?x ?y) (robot-at ?w))
    :effect (and (not(carry ?x ?y)) (box-at ?x ?w) (free ?y))
  )
)
```

# PDDL – Problem File

- Objects: rooms, boxes, and arms.

```
(:objects
  room1 room2 - room
  box1 box2 box3 box4 - box
  left right - arm
)
```

- Initial State: the robot and all boxes are at room 1.

```
(:init
  (robot-at room1)
  (box-at box1 room1)
  (box-at box2 room1)
  (box-at box3 room1)
  (box-at box4 room1)
  (free left)
  (free right)
)
```

# PDDL – Problem File

- Goal: all boxes must be at room 2.

```
(:goal
  (and (box-at box1 room2)
        (box-at box2 room2)
        (box-at box3 room2)
        (box-at box4 room2)
      )
)
```

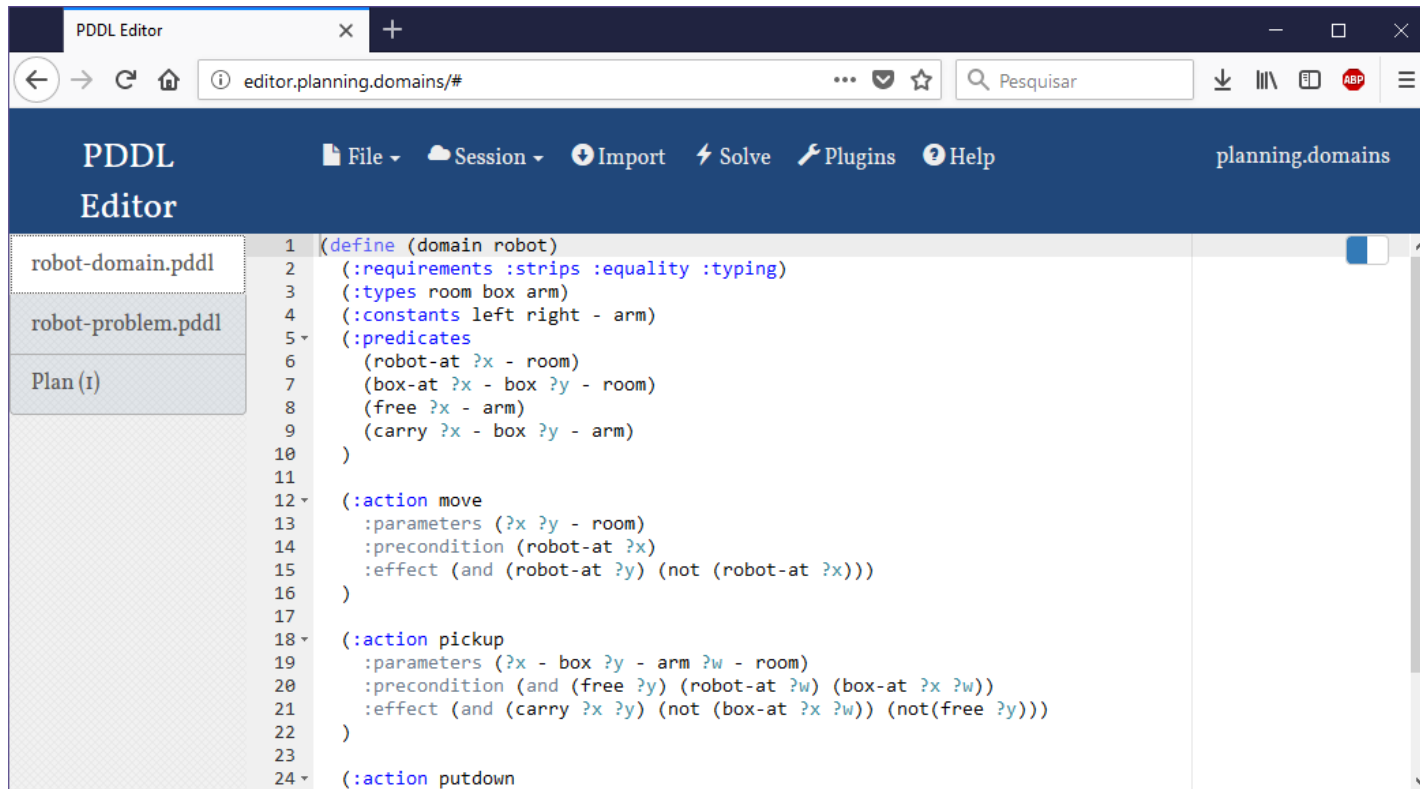
# PDDL – Problem File

```
(define (problem robot1)
  (:domain robot)
  (:objects
    room1 room2 - room
    box1 box2 box3 box4 - box
    left right - arm
  )

  (:init
    (robot-at room1)
    (box-at box1 room1)
    (box-at box2 room1)
    (box-at box3 room1)
    (box-at box4 room1)
    (free left)
    (free right)
  )

  (:goal
    (and
      (box-at box1 room2)
      (box-at box2 room2)
      (box-at box3 room2)
      (box-at box4 room2)
    )
  )
)
```

# Online PDDL Planner



```
1 (define (domain robot)
2   (:requirements :strips :equality :typing)
3   (:types room box arm)
4   (:constants left right - arm)
5   (:predicates
6     (robot-at ?x - room)
7     (box-at ?x - box ?y - room)
8     (free ?x - arm)
9     (carry ?x - box ?y - arm)
10  )
11
12  (:action move
13    :parameters (?x ?y - room)
14    :precondition (robot-at ?x)
15    :effect (and (robot-at ?y) (not (robot-at ?x))))
16  )
17
18  (:action pickup
19    :parameters (?x - box ?y - arm ?w - room)
20    :precondition (and (free ?y) (robot-at ?w) (box-at ?x ?w))
21    :effect (and (carry ?x ?y) (not (box-at ?x ?w)) (not (free ?y))))
22  )
23
24  (:action putdown
```

<http://editor.planning.domains/>



# Online PDDL Planner

- Resulting plan:

```
(pickup box1 left room1)
(move room1 room2)
(putdown box1 left room2)
(move room2 room1)
(pickup box2 left room1)
(move room1 room2)
(putdown box2 left room2)
(move room2 room1)
(pickup box3 left room1)
(move room1 room2)
(putdown box3 left room2)
(move room2 room1)
(pickup box4 left room1)
(move room1 room2)
(putdown box4 left room2)
```

# Writing Narratives as Planning Problems

- **Instead of thinking about storylines, focus on defining:**

- **Facts** (entities and relationships)

```
character(marian)
character(brian)
at(marian, castle)
love(brian, marian)
```

- **Generic events** (planning operators)

```
kill(CH1, CH2, PL)
precond: alive(CH1) ∧ alive(CH2) ∧ at(CH1, PL) ∧ at(CH2, PL)
effect: ¬alive(CH2)
```

- **Situation-Objective Rules** (authorial objectives)

```
□[(victim(V) ∧ villain(L) ∧ hero(H) ∧
    killed(V, L) ⇒ ◇ killed(V, H)]
```

# Writing Narratives as Planning Problems

- **Example 1:**

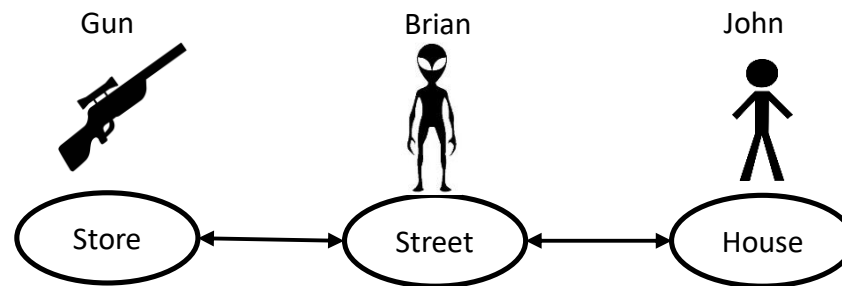
```
reduce_protection(Marian, White_Palace), go(Draco,
White_Palace), attack(Draco, White_Palace), kidnap(Draco,
Marian), go(Brian, Green_Forest), donate(Turjan, Brian),
go(Brian, Red_Castle), attack(Brian, Red_Castle),
fight(Draco, Brian), kill(Brian, Draco), free(Brian, Marian),
go(Marian, Church), go(Brian, Church), marry(Brian, Marian).
```

- **Example 2:**

```
reduce_protection(Marian, White_Palace), go(Draco,
White_Palace), attack(Draco, White_Palace), kidnap(Draco,
Marian), go(Brian, Red_Castle), go(Hoel, Red_Castle),
attack(Hoel, Red_Castle), fight(Draco, Brian), kill(Brian,
Draco), free(Hoel, Marian), go(Marian, Church), go(Hoel,
Church), marry(Hoel, Marian).
```

# Writing Narratives as Planning Problems

- Example: *“Brian wants to kill John, but he can't do much without a weapon.”*
  - The story world comprises three places: store, street and a house;
  - There is a gun at the store;
  - Brian is at the street;
  - John is at the house;



# Writing Narratives as Planning Problems

```
(define (domain simplegame)
  (:requirements :strips :equality :typing)
  (:types location character enemy weapon)
  (:predicates
    (at ?c ?l)
    (path ?l1 ?l2)
    (has ?c ?w)
    (dead ?c)
  )
  (:action go
    :parameters (?c - character ?l1 - location ?l2 - location)
    :precondition (and (at ?c ?l1) (path ?l1 ?l2))
    :effect (and (at ?c ?l2) (not (at ?c ?l1)))
  )
  (:action get
    :parameters (?c - character ?w - weapon ?l - location)
    :precondition (and (at ?c ?l) (at ?w ?l))
    :effect (and (has ?c ?w) (not (at ?w ?l)))
  )
  (:action kill
    :parameters (?c - character ?e - enemy ?w - weapon ?l - location)
    :precondition (and (at ?c ?l) (at ?e ?l) (has ?c ?w))
    :effect (and (dead ?e) (not (at ?e ?l)))
  )
)
```

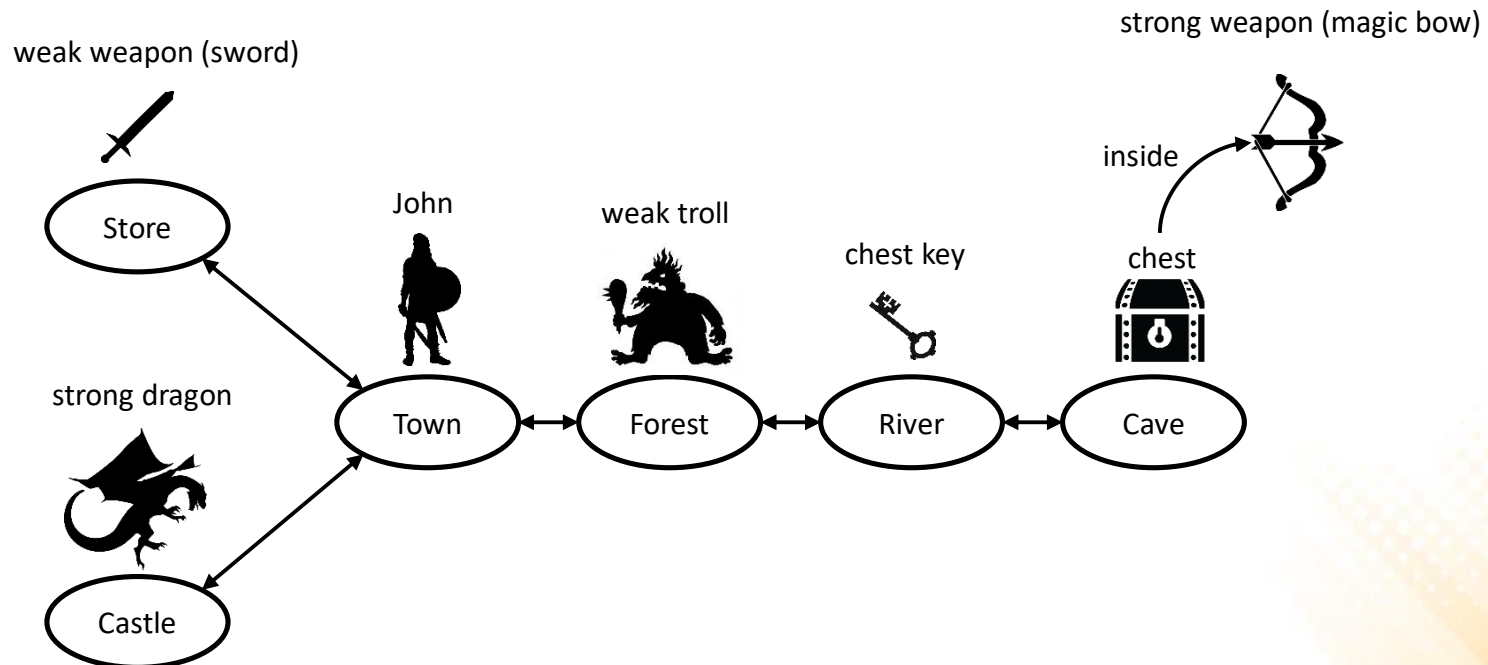
# Writing Narratives as Planning Problems

```
(define (problem npc1)
  (:domain simplegame)
  (:objects
    store street house - location
    brian - character
    john - enemy
    gun - weapon
  )
  (:init
    (at brian street)
    (at john house)
    (at gun store)
    (path store street)
    (path street store)
    (path street house)
    (path house street)
  )
  (:goal
    (and
      (dead john)
    )
  )
)
```

# Exercise 2

2) Write the PDDL domain and problem files for following story:

*“Once upon a time there was a giant dragon with a single objective in mind: attack and destroy the town’s castle. But this land is protected by a strong knight called John, who would do anything to save the castle. Unfortunately, the only weapon strong enough to defeat the dragon is a magic bow that was locked inside of a chest and buried in a cave. Recovering the magic bow is not an easy task. Anyone who joins the quest to recover it must first defeat the troll that lives in a forest around the cave and then find the key of the chest that is hidden in a river near the cave. Luckily, the troll can be easily defeated with a sword that can be bought at the town's store.”*



# Project Assignment 2

- 2) Write the interactive narrative for your project using branching networks or planning problems.
  - Remember: if your system is not designed to tell a story, the narrative should be based on a typical user interacting with the system, where the different actions that can be performed by the user represent branching points.