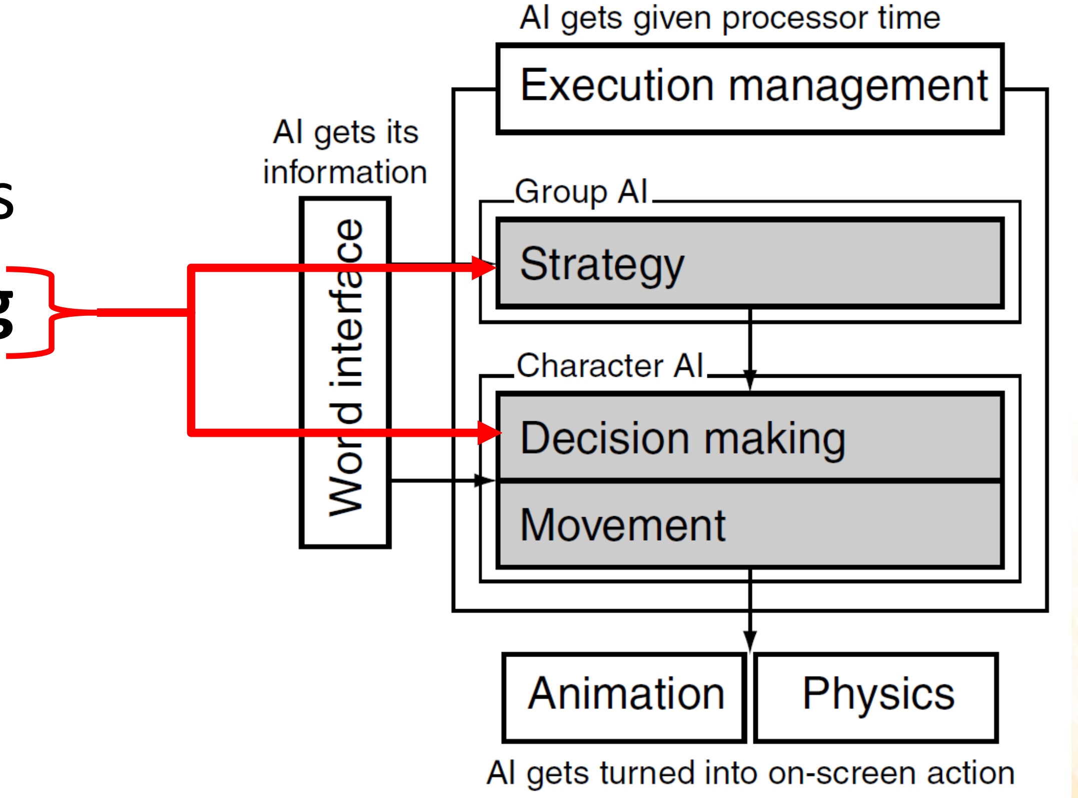
Artificial Intelligence

Lecture 04 – Automated Planning

Edirlei Soares de Lima
<edirlei.slima@gmail.com>

Game AI – Model

- Pathfinding
- Steering behaviours
- Finite state machines
- **Automated planning**
- Behaviour trees
- Randomness
- Sensor systems
- Machine learning



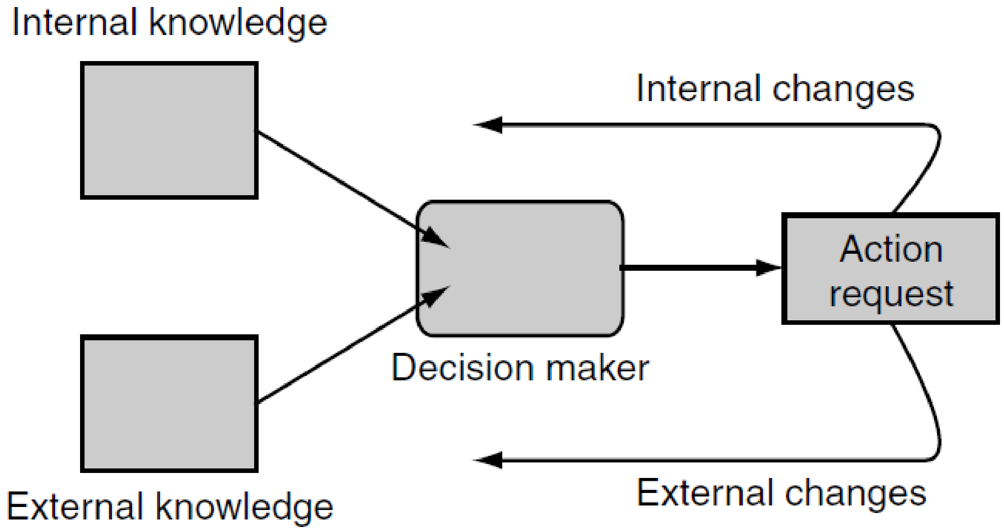
Decision Making

- In game AI, decision making is the ability of a character/agent to decide what to do.

- The agent processes a set of information that it uses to generate an action that it wants to carry out.
  - **Input:** agent’s knowledge about the world;
  - **Output:** an action request;

Decision Making

- The knowledge can be broken down into external and internal knowledge.
  - **External knowledge:** information about the game environment (e.g. characters' positions, level layout, noise direction).
  - **Internal knowledge:** information about the character's internal state (e.g. health, goals, last actions).



Goal-Oriented Behavior

- So far we have focused on reactive agents: a set of inputs is provided to the character, and an appropriate action is selected.
  - Goal-oriented behavior is an alternative approach. It adds character goals/desires to the decision making process.

- To allow an NPC to properly anticipate the effects and take advantage of sequences of actions, a planning process is required.
  - Automated Planning Techniques.

Automated Planning

- Planning is the task of finding a sequence of actions (a plan) to achieve a goal.

- **Example:**
  - Goal: `have(sword)` ∧ `at(castle)`
  - Plan: `go(dungeon), kill(enemy), get(key), go(forest), open(chest, key), get(sword), go(castle).`

- Plan-based agent process:
  1) Formulate a goal;
  2) Find a plan;
  3) Execute the plan;

Automated Planning

- 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 problem elements**:
  - Initial State;
  - Actions (with preconditions and effects);
  - Goal;

Planning Problem

- Each <u>state</u> is represented as a conjunction of predicates.
  - Example: `At(Truck1, Melbourne)` ∧ `At(Truck2, Sydney).`
  - <u>Closed-world assumption</u>: any predicates that are not mentioned are false.

- <u>Actions</u> are described by a set of action schemas with <u>parameters</u>, <u>preconditions</u>, and <u>effects</u>.
  - Example:

```
Action(
  Fly(p, f, t),
  PRECOND: At(p, f) ∧ Plane(p) ∧ Airport(f) ∧ Airport(t)
  EFFECT: ¬At(p, f) ∧ At(p, t)
)
```

Planning Problem

- The precondition defines the states in which the action can be executed.

- Example:

```
Action(
  Fly(p, f, t),
  PRECOND: At(p, f) ∧ Plane(p) ∧ Airport(f) ∧ Airport(t)
  EFFECT: ¬At(p, f) ∧ At(p, t)
)
```

  - **Initial State:** At(C1, SFO) ∧ At(C2, JFK) ∧ At(P1, SFO) ∧ At(P2, JFK) ∧ Cargo(C1) ∧ Cargo(C2) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport (JFK) ∧ Airport (SFO)

  - The Fly action can be instantiated as Fly(P1, SFO, JFK) or as Fly(P2, JFK, SFO).

Planning Problem

- The effect defines the result of executing the action.

- Example:

```
Action(
  Fly(p, f, t),
  PRECOND: At(p, f) ∧ Plane(p) ∧ Airport(f) ∧ Airport(t)
  EFFECT: ¬At(p, f) ∧ At(p, t)
)
```

  - **Initial State:** At(C1, SFO) ∧ At(C2, JFK) ∧ At(P1, SFO) ∧ At(P2, JFK) ∧ Cargo(C1) ∧ Cargo(C2) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport (JFK) ∧ Airport (SFO)

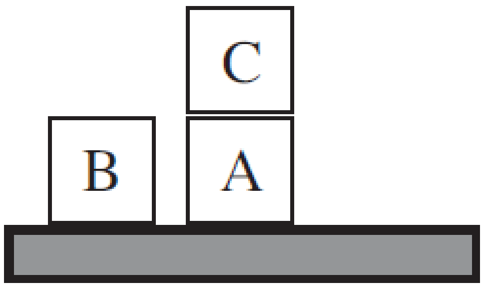
  - Negative predicates are removed from the resulting state (e.g. `¬At(p, f)`);
  - Positive predicates are added to the resulting state (e.g. `At(p, t)`);

Example – Air Cargo Transport

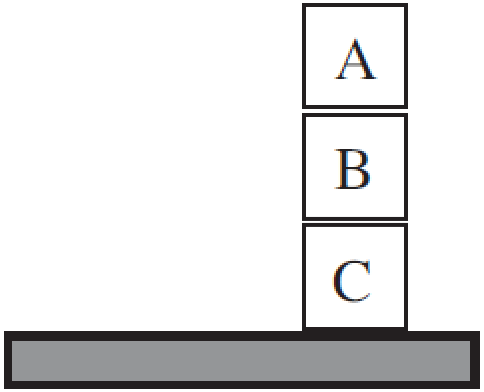
```
Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(P1, SFO) ∧ At(P2, JFK) ∧
   Cargo(C1) ∧ Cargo(C2) ∧ Plane(P1) ∧ Plane(P2) ∧
   Airport (JFK) ∧ Airport (SFO))

Goal(At(C1, JFK) ∧ At(C2, SFO))

Action(
  Load(c, p, a),
  PRECOND: At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
  EFFECT: ¬At(c, a) ∧ In(c, p)
)
Action(
  Unload(c, p, a),
  PRECOND: In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
  EFFECT: At(c, a) ∧ ¬In(c, p)
)
Action(
  Fly(p, f, t),
  PRECOND: At(p, f) ∧ Plane(p) ∧ Airport(f) ∧ Airport(t)
  EFFECT: ¬At(p, f) ∧ At(p, t)
)
```

Example – Blocks World

```
Init(On(A, Table) ∧ On(B, Table) ∧ On(C, A) ∧
     Block(A) ∧ Block(B) ∧ Block(C) ∧ Clear(B) ∧
     Clear(C))

Goal(On(A,B) ∧ On(B,C))

Action(
  Move(b, x, y),
  PRECOND: On(b, x) ∧ Clear(b) ∧ Clear(y) ∧
           Block(b) ∧ Block(y) ∧ (b ≠ x) ∧
           (b ≠ y) ∧ (x ≠ y),
  EFFECT: On(b, y) ∧ Clear(x) ∧ ¬On(b, x) ∧
          ¬Clear(y)
)
Action(
  MoveToTable(b, x),
  PRECOND: On(b, x) ∧ Clear(b) ∧ Block(b) ∧
           (b ≠ x),
  EFFECT: On(b, Table) ∧ Clear(x) ∧ ¬On(b, x)
)
```



Start State

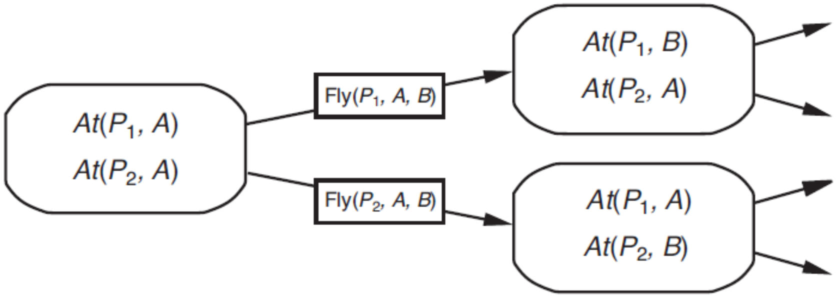


Goal State

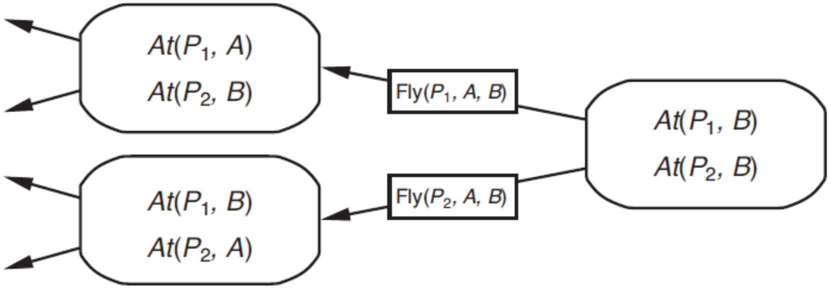
Planning Algorithms

- The description of a planning problem defines a search problem: we can search from the initial state looking for a goal.

- Planning approaches:
  - Progressive: forward state-space search;

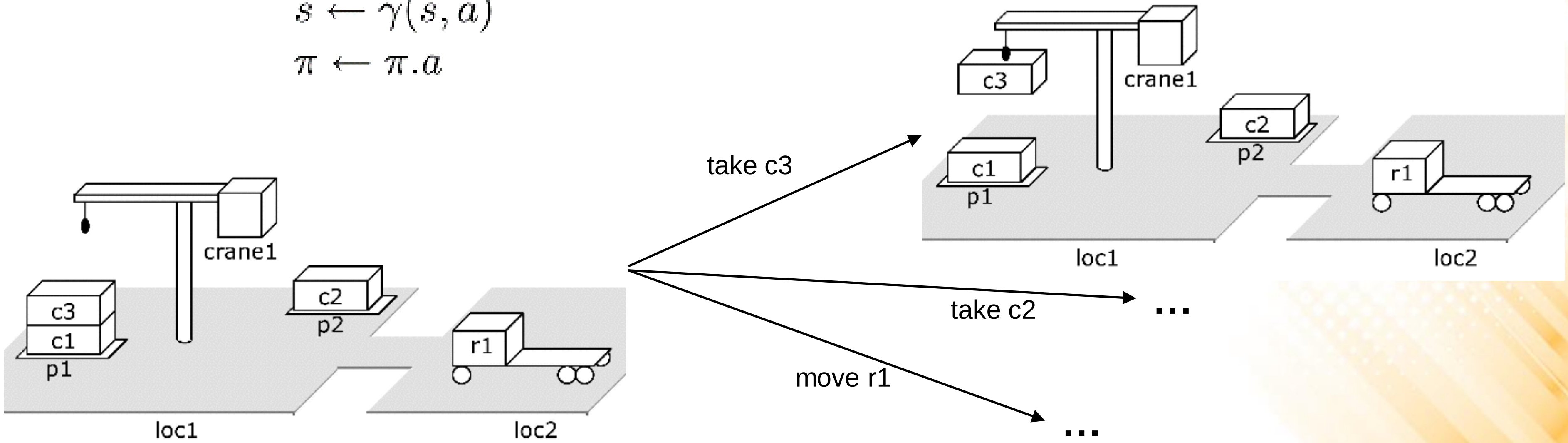


  - Regressive: backward relevant-states search;



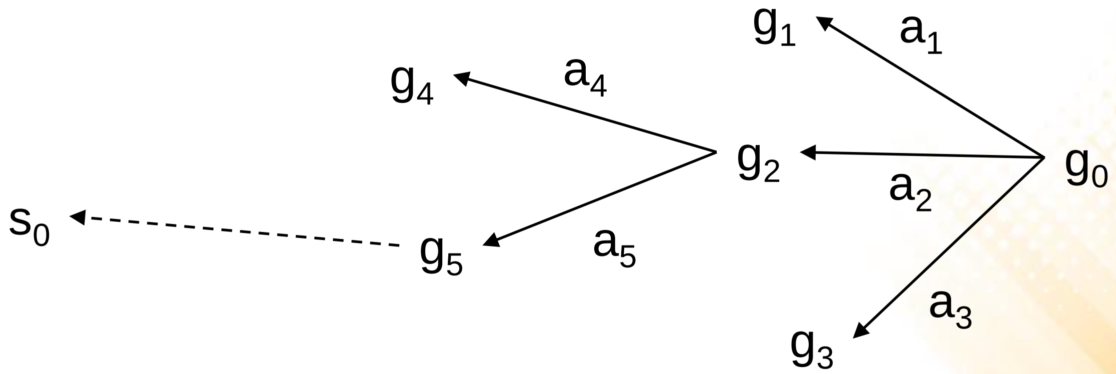
Forward State-Space Search

Forward-search(O, s_0, g)
    $s \leftarrow s_0$
    $\pi \leftarrow$ the empty plan
    loop
        if s satisfies g then return π
        $E \leftarrow \{a | a$ is a ground instance an operator in O,
                and $\mathrm{precond}(a)$ is true in $s\}$
        if $E = \emptyset$ then return failure
        nondeterministically choose an action $a \in E$
        $s \leftarrow \gamma(s, a)$
        $\pi \leftarrow \pi.a$



Backward Relevant-States Search

Backward-search(O, s_0, g)
  $\pi \leftarrow$ the empty plan
  loop
    if s_0 satisfies g then return π
    $A \leftarrow \{a | a$ is a ground instance of an operator in O
      and $\gamma^{-1}(g, a)$ is defined$\}$
    if $A = \emptyset$ then return failure
    nondeterministically choose an action $a \in A$
    $\pi \leftarrow a.\pi$
    $g \leftarrow \gamma^{-1}(g, a)$



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:**

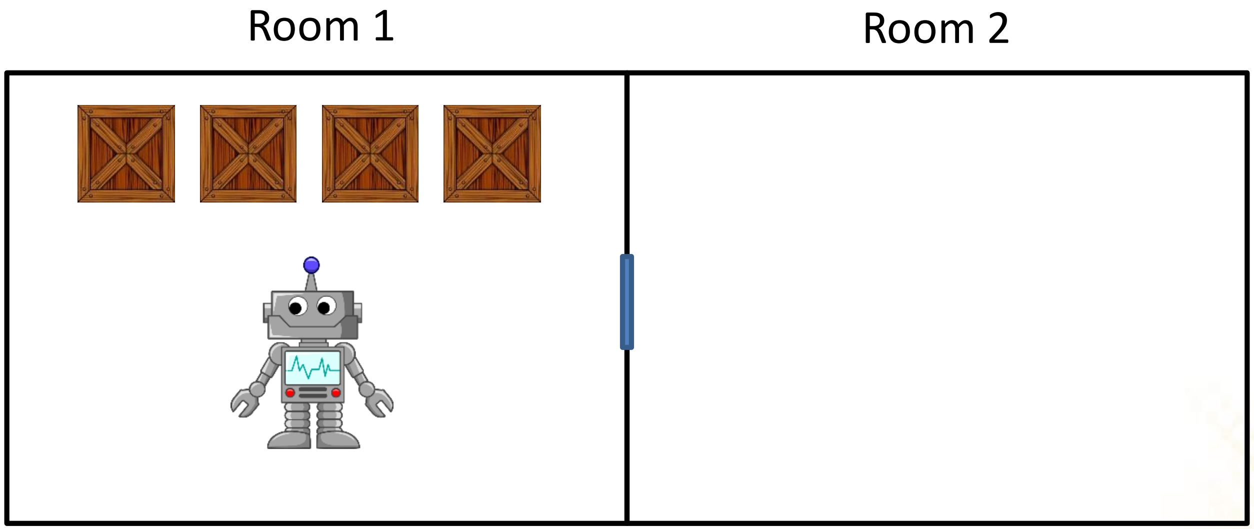
```
(define (domain <domain name>)
      (:requirements :strips :equality :typing)
      (:types <list of types>)
      (:constants <list of constants>)
      <PDDL code for predicates>
      <PDDL code for first action>
      [...]
      <PDDL code for last action>
)
```

- **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)
    )
  )
)
```

PDDL – Planners

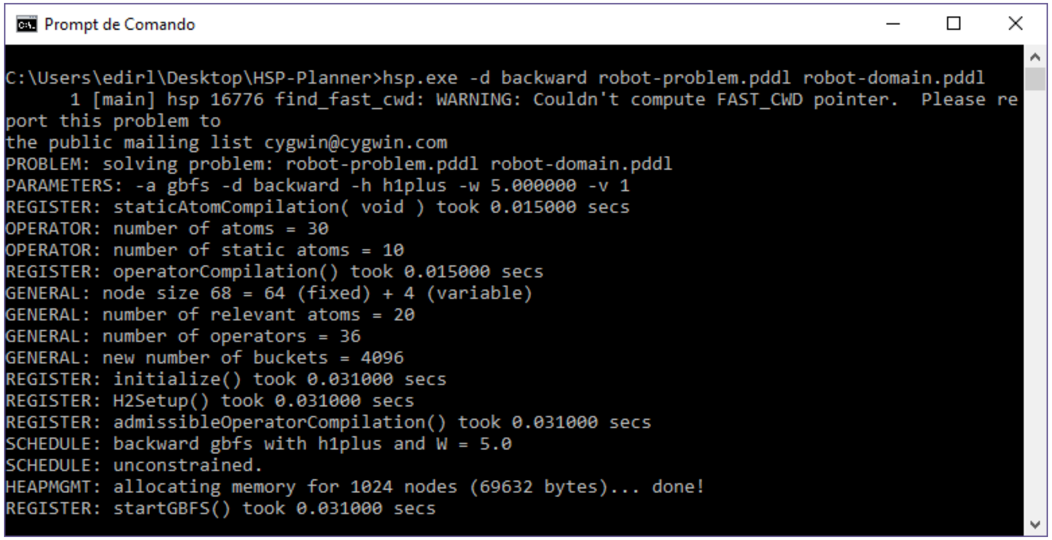
- HSP Planner - https://github.com/bonetblai/hsp-planners
  - Heuristic Search Planner;
  - Compiled version for windows (cygwin): http://edirlei.3dgb.com.br/aulas/ia_2013_1/HSP-Planner.zip

- Online PDDL Planner:
  - Editor: http://editor.planning.domains/
  - Remote API: http://solver.planning.domains/

HSP Planner

- **Executing the planner:**
  - hsp.exe robot-problem.pddl robot-domain.pddl

- **Extra parameters:**
  - Search direction: -d backward ou forward
  - Search algorithm: -a bfs ou gbfs



HSP Planner

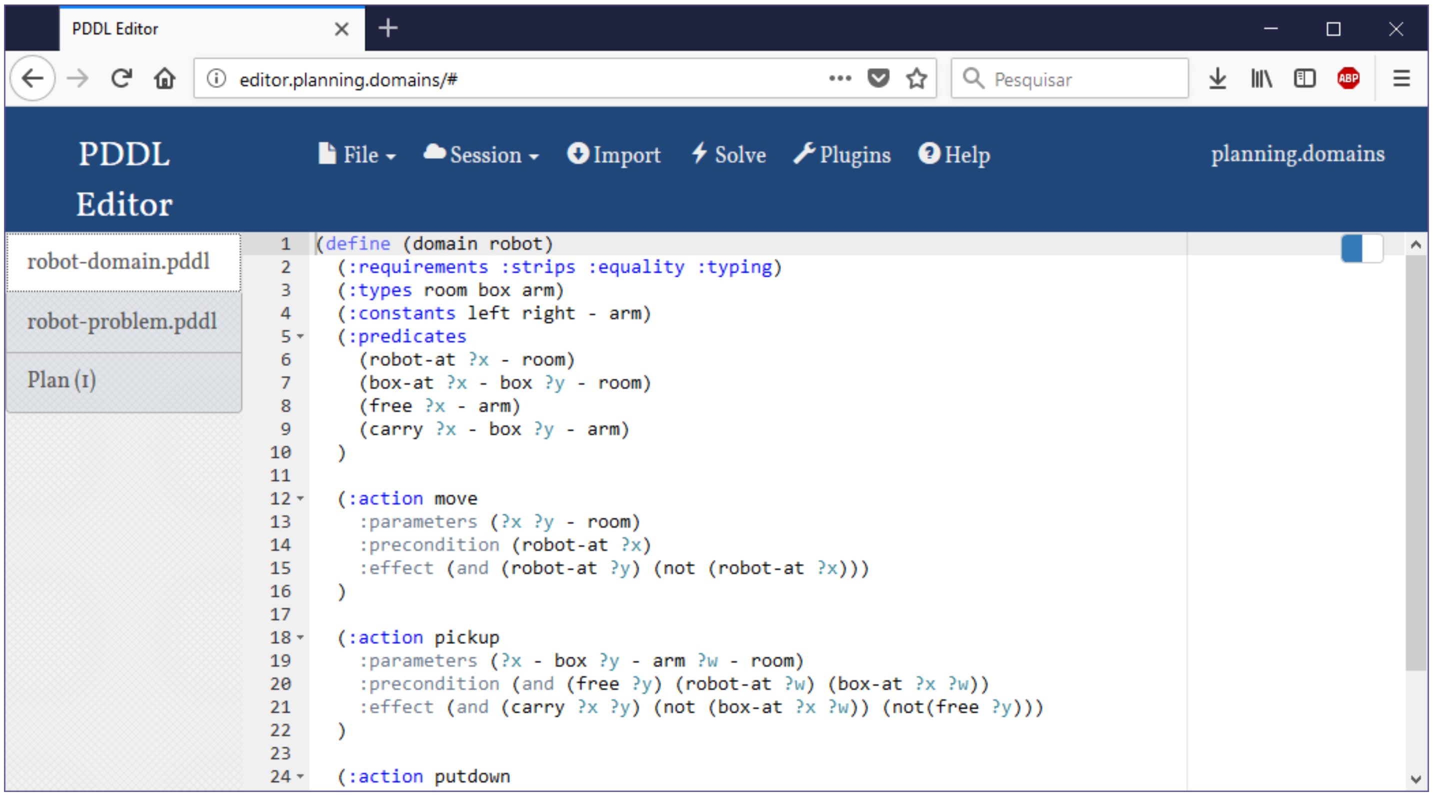
- Forward search:

```
(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)
(PICKUP BOX4 RIGHT ROOM1)
(MOVE ROOM1 ROOM2)
(PUTDOWN BOX3 LEFT ROOM2)
(PUTDOWN BOX4 RIGHT ROOM2)
```

- Backward search:

```
(PICKUP BOX4 RIGHT ROOM1)
(PICKUP BOX3 LEFT ROOM1)
(MOVE ROOM1 ROOM2)
(PUTDOWN BOX4 RIGHT ROOM2)
(PUTDOWN BOX3 LEFT ROOM2)
(MOVE ROOM2 ROOM1)
(PICKUP BOX2 RIGHT ROOM1)
(PICKUP BOX1 LEFT ROOM1)
(MOVE ROOM1 ROOM2)
(PUTDOWN BOX2 RIGHT ROOM2)
(PUTDOWN BOX1 LEFT ROOM2)
```

Online PDDL Planner



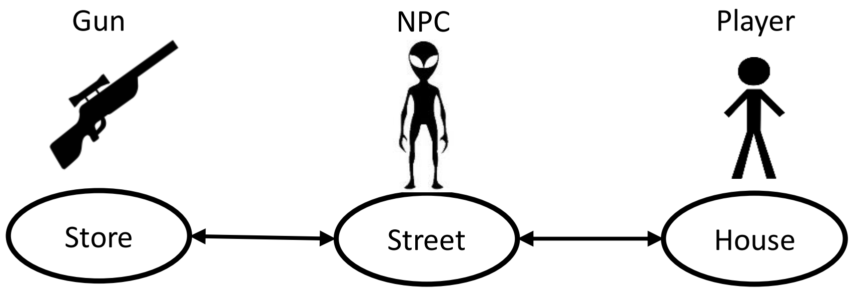
Online PDDL Planner

- 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)
```

PDDL – Simple Game Situation

- *“The objective of the NPC is to kill the player, but he can't do much without a weapon.”*
    - The game world comprises three places: store, street and a house;
    - There is a gun at the store;
    - The NPC is at the street;
    - The player is at the house;



PDDL – Simple Game Situation

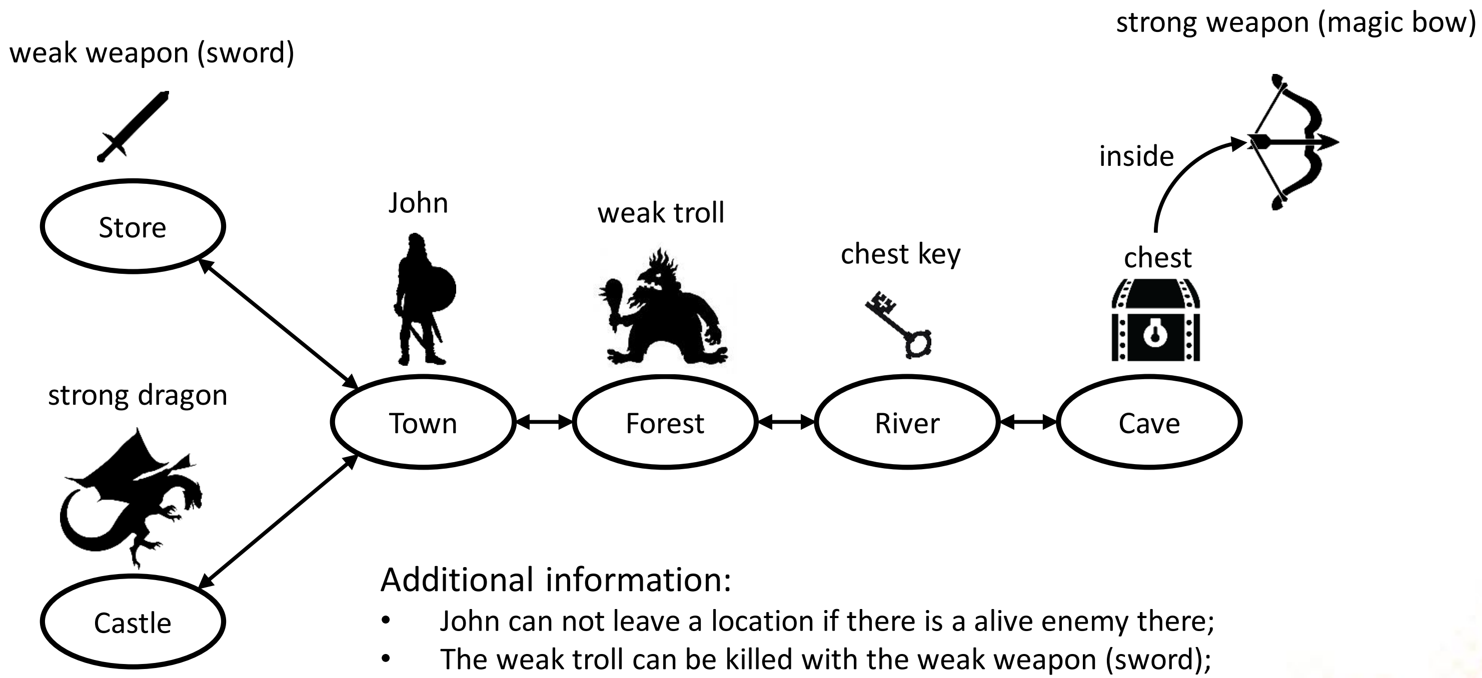
```
(define (domain simplegame)
  (:requirements :strips :equality :typing)
  (:types location character enemy weapon)
  (:predicates
    (at ?c ?l)
    (path ?l1 ?l2)
    (has ?c ?w)
    (dead ?c)
  )
  (:action go
    :parameters (?c - character ?l1 - location ?l2 - location)
    :precondition (and (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)))
  )
)
```

PDDL – Simple Game Situation

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

Exercise 1

1) Implement the PDDL domain and problem files to solve the following problem: “*A giant dragon is attacking the castle and John must find a way to kill the dragon!*”



Additional information:

- John can not leave a location if there is a alive enemy there;
- The weak troll can be killed with the weak weapon (sword);
- The chest is closed. It can be opened with the chest key;
- There is a strong weapon inside of the chest (magic bow);
- The dragon can only be killed with a strong weapon (the magic bow);

Automated Planning in Unity

- The best way to add automated planning to a Unity project is by implementing the planning algorithm directly in Unity.
  - Starting point: C# PDDL Parser - https://github.com/sunsided/pddl

- Alternatively, we can use a modified version of the HSP Planner (written in C) as a standard alone application that can be executed by an Unity script to generate plans.
  - http://edirlei.3dgb.com.br/aulas/game-ai/HPS-Planner-Unity.zip
  - Not an efficient solution. Use it only for prototyping purposes.

- Another option: use the online planning service API:
  - http://solver.planning.domains/
  - Limitations: internet connection, speed, server overload, ...

Automated Planning in Unity

- Executing the HSP Planner in Unity:

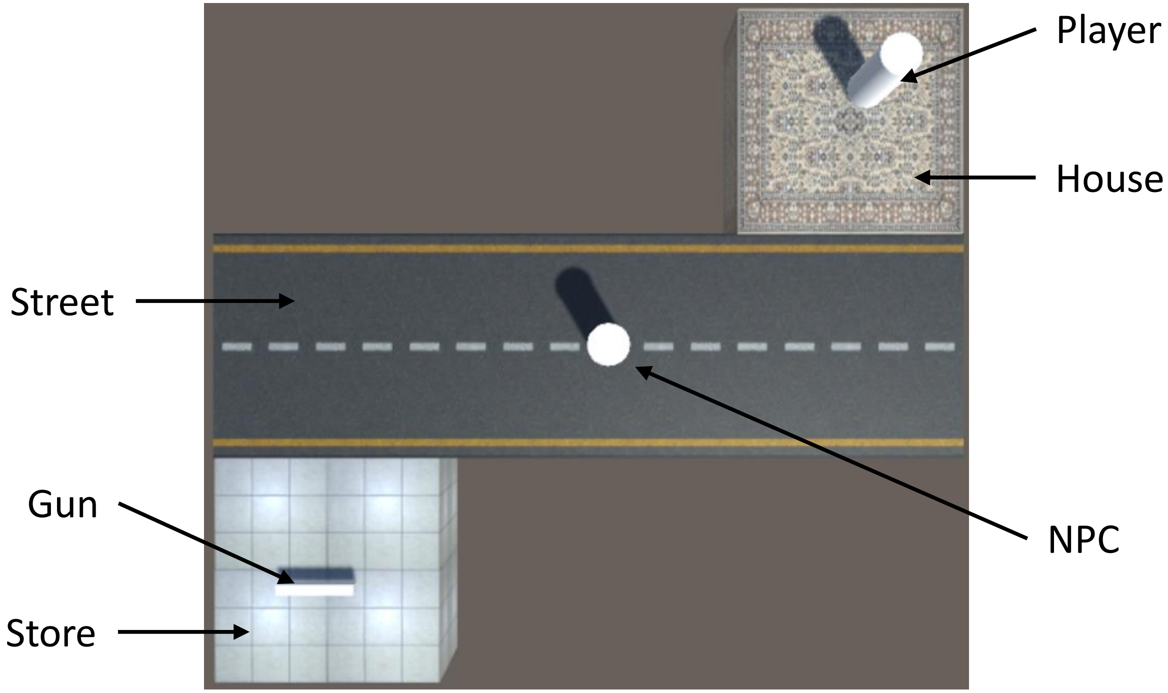
```
using System.Diagnostics;
...
try{
  Process plannerProcess = new Process();
  plannerProcess.StartInfo.FileName = "Planner/hsp2.exe";
  plannerProcess.StartInfo.CreateNoWindow = true;
  plannerProcess.StartInfo.Arguments = "Planner/game-problem.pddl
                                         Planner/game-domain.pddl";
  plannerProcess.StartInfo.UseShellExecute = false;
  plannerProcess.StartInfo.RedirectStandardOutput = true;
  plannerProcess.Start();
  plannerProcess.WaitForExit();
  while (!plannerProcess.StandardOutput.EndOfStream){
    UnityEngine.Debug.Log(plannerProcess.StandardOutput.ReadLine());
  }
}catch (System.Exception e){
  UnityEngine.Debug.Log(e);
}
```

Relative path of the HSP exe in the project folder.

Processes the plan actions individually.

Automated Planning in Unity - Example

- **Simple Game Situation Example**: *“The objective of the NPC is to kill the player, but he can't do much without a weapon.”*



```
public class PlanAction {
  public string name;
  public List<string> parameters;
  public Status status;

  public PlanAction(string action){
    string temp = "";
    name = "";
    parameters = new List<string>();
    foreach (char c in action){
      if (c == ' '){
        if (name.Equals(""))
          name = temp;
        else
          parameters.Add(temp);
        temp = "";
      }
      else if (c == ')')
        parameters.Add(temp);
      else if (c != '(')
        temp += c;
    }
    status = Status.Ready;
  }
}
```

Class to store and interpret planner actions.

```
public enum Status { Ready,
                     Executing,
                     Completed
};
```

```
public class NPCPlanner : MonoBehaviour {
  private List<PlanAction> plan;
  private int currentAction;
  private NavMeshAgent agent;
  public WaypointInfo[] waypoints;
  void Start(){
    plan = new List<PlanAction>();
    agent = GetComponent<NavMeshAgent>();
    currentAction = 0;
    try{
      Process planner = new Process();
      planner.StartInfo.FileName = "Planner/hsp2.exe";
      planner.StartInfo.CreateNoWindow = true;
      planner.StartInfo.Arguments = "Planner/game-problem.pddl
                                     Planner/game-domain.pddl";
      planner.StartInfo.UseShellExecute = false;
      planner.StartInfo.RedirectStandardOutput = true;
      planner.Start();
      planner.WaitForExit();
      while (!planner.StandardOutput.EndOfStream){
        plan.Add(new PlanAction(planner.StandardOutput.ReadLine()));
      }
    }catch (System.Exception e){
            UnityEngine.Debug.Log(e);
    }
  }
```

```
[System.Serializable]
public struct WaypointInfo
{
    public string name;
    public Transform waypoint;
}
```

```
void Update(){
  if (currentAction < plan.Count){
    if (plan[currentAction].status == Status.Ready){
      DoAction(plan[currentAction]);
    }
    if (plan[currentAction].status == Status.Executing){
      CheckAction(plan[currentAction]);
    }
    if (plan[currentAction].status == Status.Completed){
      currentAction++;
    }
  }
}
void DoAction(PlanAction action){
  if (action.name.Equals("GO")){
    agent.destination = GetWaypoint(action.parameters[2]);
    action.status = Status.Executing;
  }
  else if (action.name.Equals("GET")){
    Destroy(GameObject.FindGameObjectWithTag(action.parameters[1]));
    action.status = Status.Executing;
  }
  else if (action.name.Equals("KILL")){
    Destroy(GameObject.FindGameObjectWithTag(action.parameters[1]));
    action.status = Status.Executing;
  }
}
```

Just an example. Usually you should play an animation.

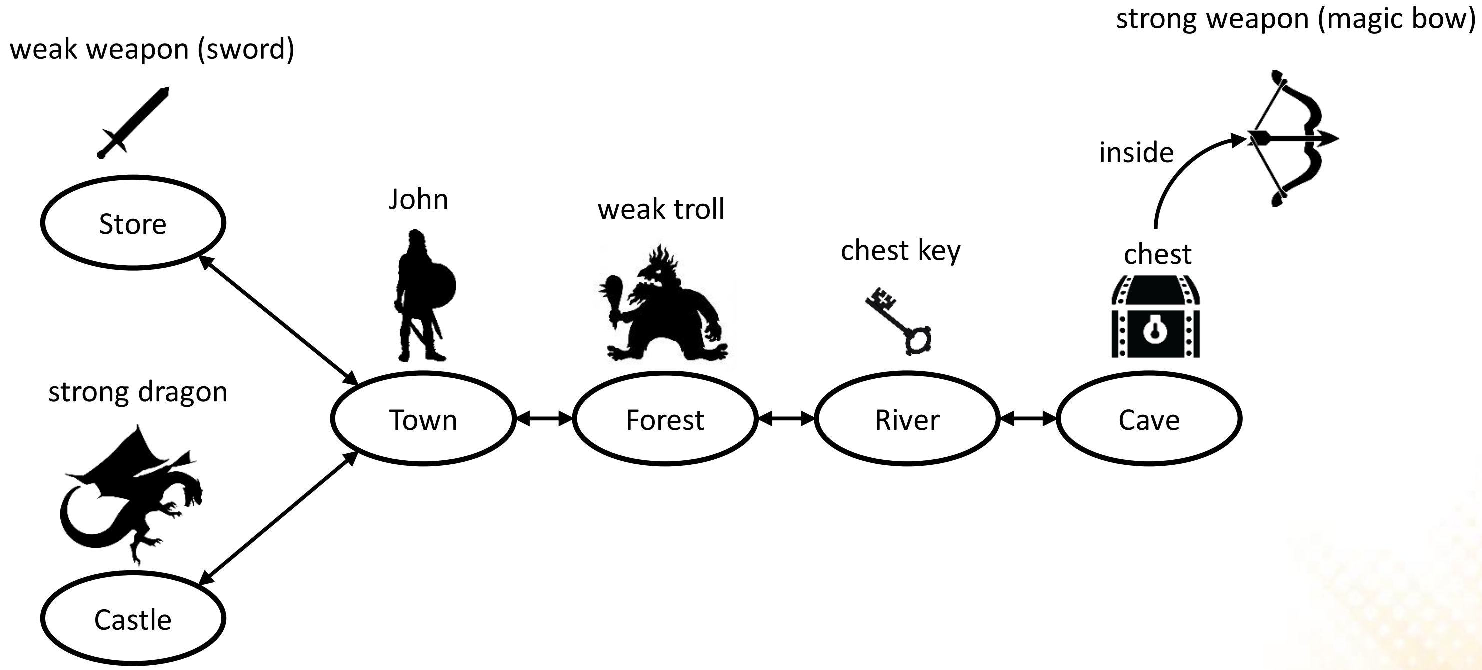
```
  void CheckAction(PlanAction action){
    if (action.name.Equals("GO")){
      if (IsAtDestionation())
        action.status = Status.Completed;
    }
    else if (action.name.Equals("GET")){
      action.status = Status.Completed;
    }
    else if (action.name.Equals("KILL")){
      action.status = Status.Completed;
    }
  }

  Vector3 GetWaypoint(string name){
    foreach (WaypointInfo wp in waypoints){
      if (wp.name.Equals(name))
        return wp.waypoint.position;
     }
    return Vector3.zero;
  }

  public bool IsAtDestionation(){
    ...
  }
}
```

Usually you need to wait until the animation ends.

Same function implemented in last lecture.

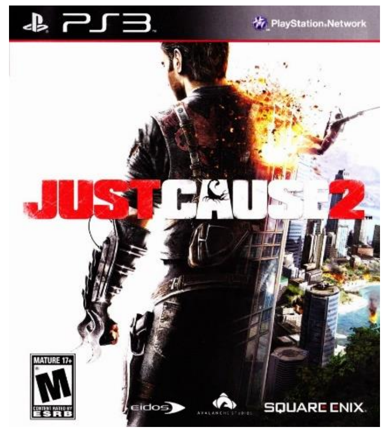
Exercise 2

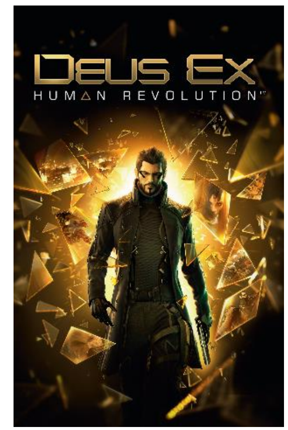
2) Create a scene to represent the world specified in Exercise 1. Then, integrate the HSP Planner in the project and implement the actions of the NPC John to execute the generated plan.



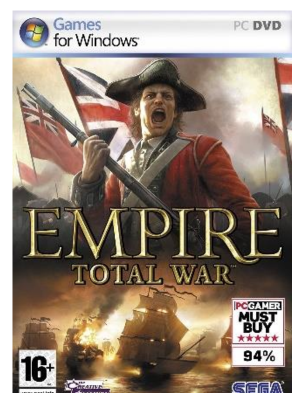
Automated Planning in Games

- Games that are know for using planning algorithms:
  - STRIPS-based action planning:





  - HTN-based action planning:

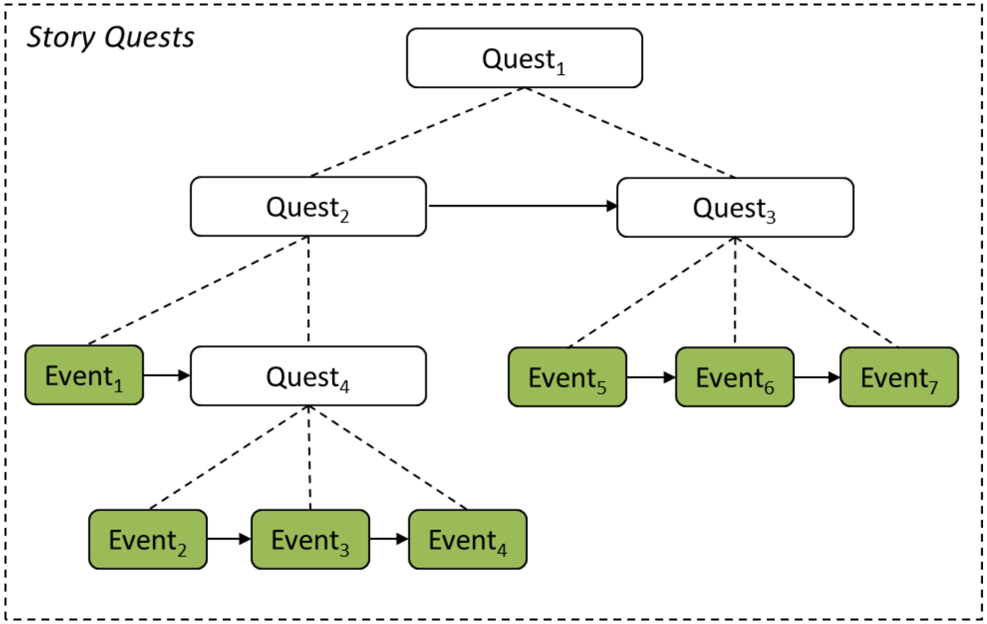


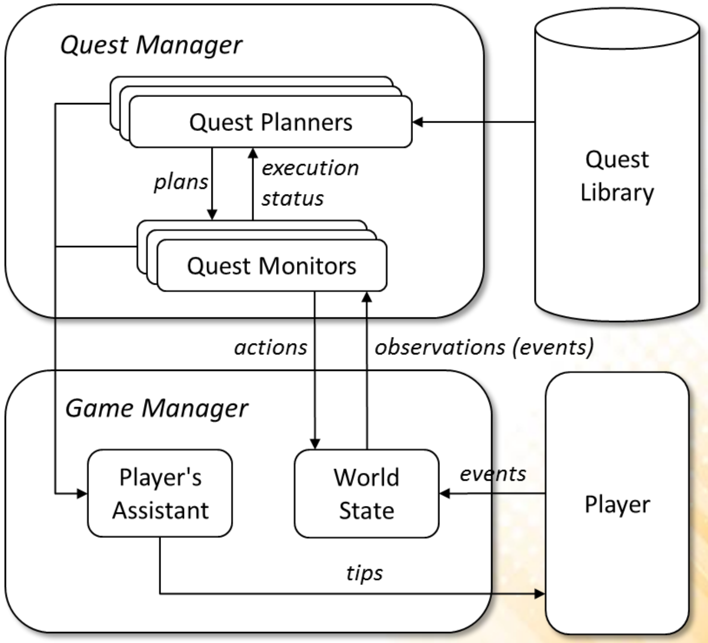
Automated Planning in Games

- There are many possible applications for automated planning in games:
  - Planning NPC actions;
  - Strategy planning;
  - Design, test, and evaluate puzzles;
  - Quest generation;
  - Interactive storytelling;

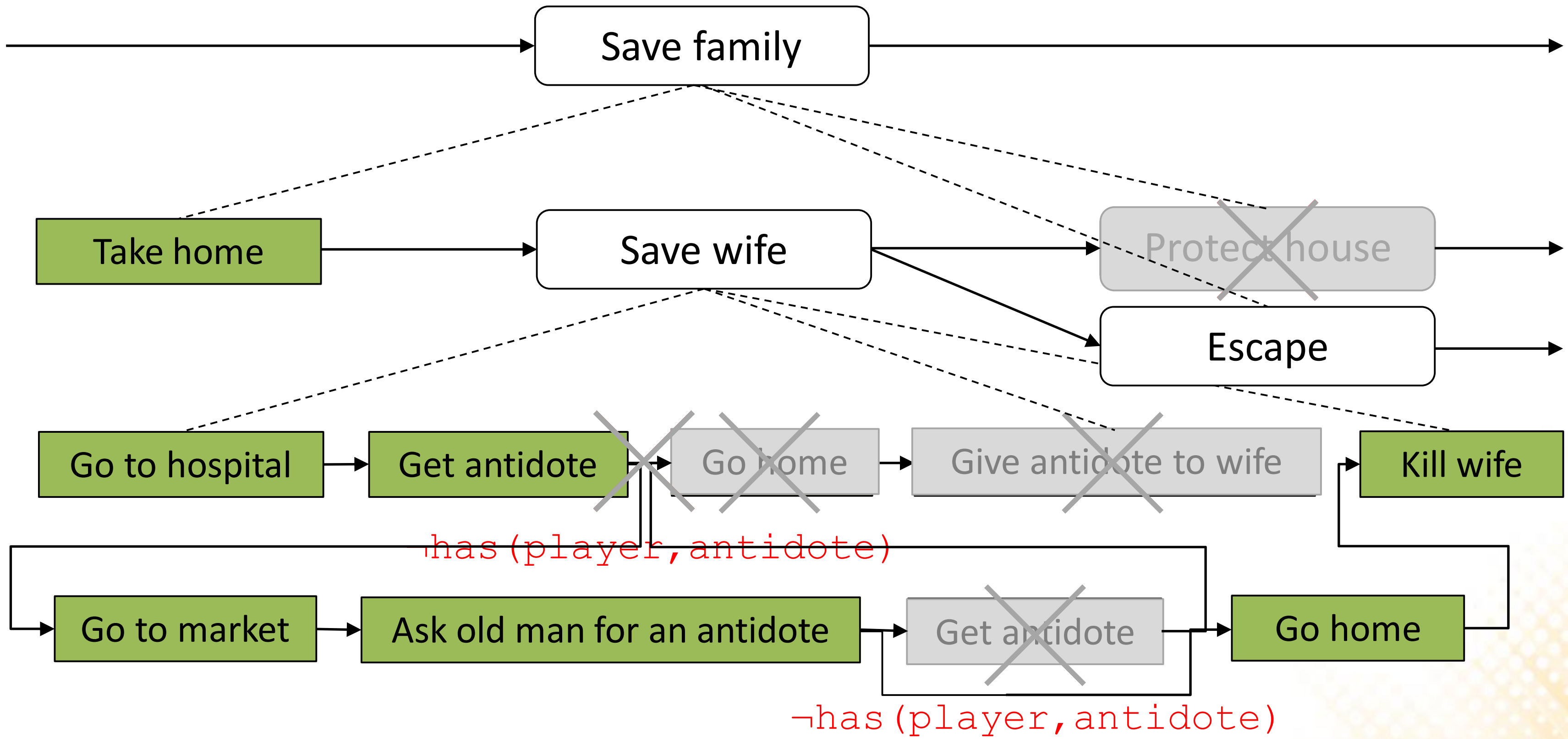
Hierarchical Generation of Dynamic and Nondeterministic Quests

- A combination of several <u>story-related quests</u> can be used to create complex narratives. The structure of the game's narrative can be represented as a <u>hierarchy of quests</u>.
  - Lima, E.S. Feijó, B., and Furtado, A.L. **Hierarchical Generation of Dynamic and Nondeterministic Quests in Games**. International Conference on Advances in Computer Entertainment Technology (ACE 2014).

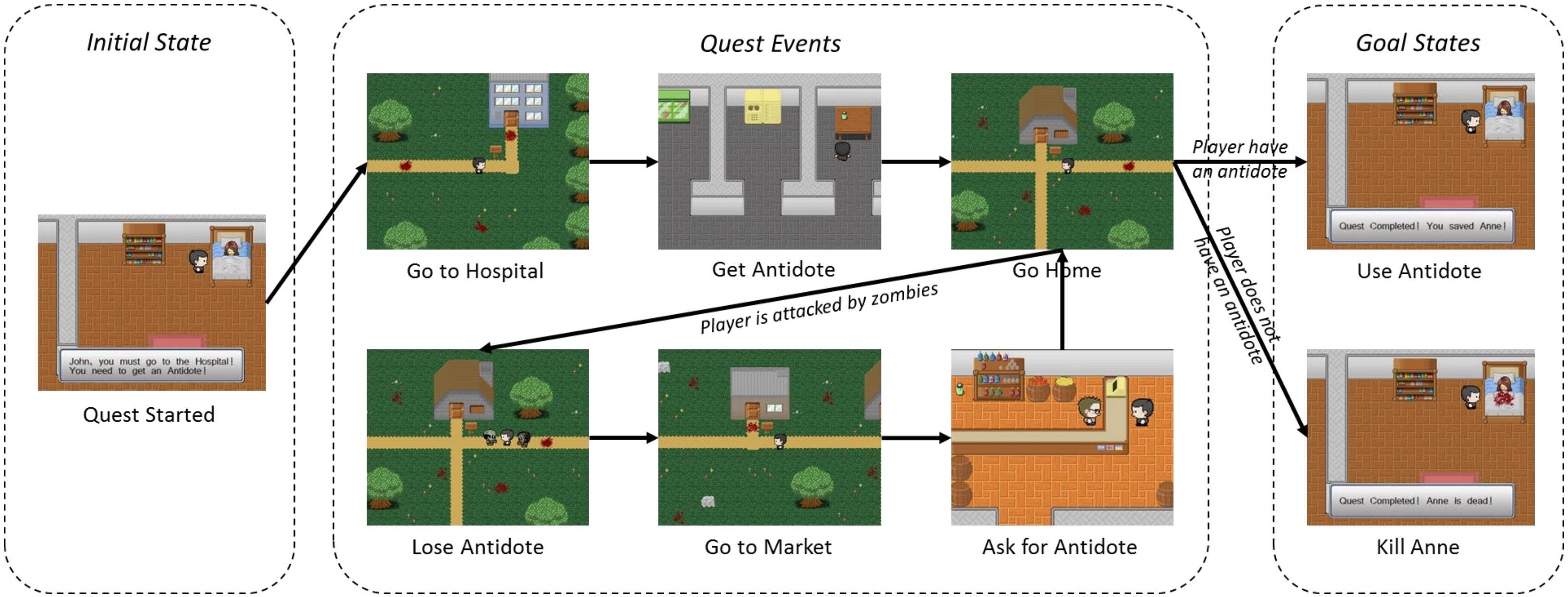




Hierarchical Generation of Dynamic and Nondeterministic Quests



Hierarchical Generation of Dynamic and Nondeterministic Quests

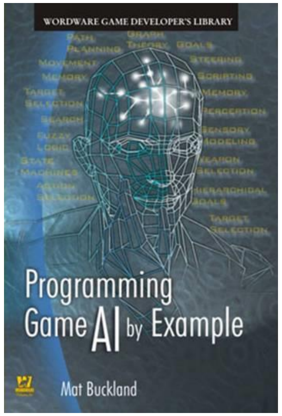


Publications:

- Lima, E.S. Feijó, B., and Furtado, A.L. Hierarchical Generation of Dynamic and Nondeterministic Quests in Games. International Conference on Advances in Computer Entertainment Technology, 2014.
- Lima, E.S. Feijó, B., and Furtado, A.L. Player Behavior Modeling for Interactive Storytelling in Games. XV Brazilian Symposium on Computer Games and Digital Entertainment, 2016 [*Best Paper Award*].
- Lima, E.S. Feijó, B., and Furtado, A.L. Player Behavior and Personality Modeling for Interactive Storytelling in Games. Entertainment Computing, Volume 28, December 2018, p. 32-48, 2018.

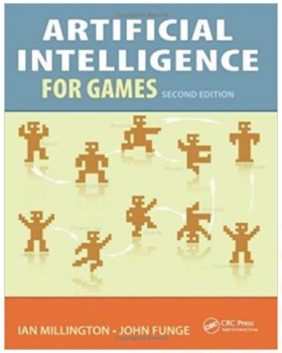
Further Reading

- Buckland, M. (2004). **Programming Game AI by Example**. Jones & Bartlett Learning. ISBN: 978-1-55622-078-4.



  – **Chapter 9: Goal-Driven Agent Behavior**

- Millington, I., Funge, J. (2009). **Artificial Intelligence for Games** (2nd ed.). CRC Press. ISBN: 978-0123747310.



  – **Chapter 5.7: Goal-Oriented Behavior**

Further Reading

- Three States and a Plan: The A.I. of F.E.A.R: http://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf

- HTN Planning in Transformers: Fall of Cybertron: https://aiandgames.com/cybertron-intel/

- Planning in Games: An Overview and Lessons Learned: http://aigamedev.com/open/review/planning-in-games/

- Goal-Oriented Action Planning (GOAP): http://alumni.media.mit.edu/~jorkin/goap.html