INTRODUCTION TO AI
STrips PLANNING

.. and Applications to Video-games!
Course overview

- Lecture 1: Game-inspired competitions for AI research, AI decision making for non-player characters in games
- Lecture 2: STRIPS planning, state-space search
- Lecture 3: Planning Domain Definition Language (PDDL), using an award winning planner to solve Sokoban
- Lecture 4: Planning graphs, domain independent heuristics for STRIPS planning
- Lecture 5: Employing STRIPS planning in games: SimpleFPS, iThinkUnity3D, SmartWorkersRTS
- Lecture 6: Planning beyond STRIPS
Artificial Intelligence and Video Games

- Video Games:
  - Finite State Machines
  - Decision Diagrams
  - Behavior Trees
  - Goal Oriented Action Planning

- Academic AI on agents:
  - Knowledge representation, First-order logic, Classical planning, Planning with preferences, …
  - Belief-Desire-Intention architecture, Agent-based programming, …
  - Probabilistic reasoning, Bayesian networks, Utility theory, Markov Decision Processes, …
Game engine:

- C++

- Creates game-world objects with \((x, y, z)\) coordinates and calculates what happens to them on every frame

- E.g., a crate is up in the air on frame 1. On frame 2 the game engine will calculate the new position, etc
Artificial Intelligence and Video Games

- **Game engine:**
  - C++
  - Creates game-world objects with \((x,y,z)\) coordinates and calculates what happens to them on every frame
  - E.g., a crate is up in the air on frame 1. On frame 2 the game engine will calculate the new position, etc
  - Same for **non-player characters** (NPCs)!
Finite State Machines (FSMs)

- Video Games:
  - Finite State Machines
  - Decision Diagrams
  - Behavior Trees
  - Goal Oriented Action Planning

- Academic AI on agents:
  - Knowledge representation, First-order logic, Classical planning, Planning with preferences, …
  - Belief-Desire-Intention architecture, Agent-based programming, …
  - Probabilistic reasoning, Bayesian networks, Utility theory, Markov Decision Processes, …
Finite State Machines (FSMs)

- Recognize a formal language

![Finite State Machine Diagram]

The diagram illustrates a finite state machine with states labeled as 'a', 'b', 'c', 'd', 'e', and 'f'. The transitions between states are labeled with '0' and '1' for input symbols in the language.
Finite State Machines (FSMs)

- NPC behavior based on high-level states

- On Guard
  - See small enemy
  - See big enemy
  - Escaped

- Fight
  - Energy OK
  - Losing fight

- Run away
Finite State Machines (FSMs)

- Traditionally one of the first techniques for NPC behavior
- Very simple to understand
- Very simple to implement
  - E.g., directly using if-then-else statements
int NPC::think()
{
    if (state==ONGUARD && seeSmallEnemy()){
        state=FIGHT;
        makeScarySound();
    }
    else if (state==FIGHT && energy>30){
        ...
    }
    else if ...
Finite State Machines (FSMs)

- Let’s see some code from a commercial game

- HL2-SDK, npc_BaseZombie.cpp
- lines 1828-1870

```cpp
switch ( m_NPCState )
{
    case NPC_STATE_COMBAT:
    ...
    case NPC_STATE_ALERT:
    ...
}
```
Finite State Machines (FSMs)

- Traditionally one of the first techniques for NPC behavior
- Very simple to understand
- Very simple to implement
  - E.g., directly using if-then-else statements
- Separation between the work of the programmer and the game designers
- But also simplistic in the behaviors that can be expressed...
Behavior Trees (BTs)

- Video Games:
  - Finite State Machines
  - Decision Diagrams
  - Behavior Trees
  - Goal Oriented Action Planning

- Academic AI on agents:
  - Knowledge representation, First-order logic,
    Classical planning, Planning with preferences, …
  - Belief-Desire-Intention architecture, Agent-based
    programming, …
  - Probabilistic reasoning, Bayesian networks,
    Utility theory, Markov Decision Processes, …
Behavior Trees (BTs)

- NPC behavior based on more refined conditions and strategies
  - **Tasks** have a common basic structure: they are given CPU time to do something and return success or failure
  - Leaf tasks: check a condition or execute some code
  - Composite tasks: return value depend on child tasks
Behavior Trees (BTs)

- NPC behavior based on more refined conditions and strategies
  - Tasks have a common basic structure: they are given CPU time to do something and return success or failure
  - **Leaf tasks:** check a condition or execute some code
  - Composite tasks: return value depend on child tasks

E.g., succeed if the door in front of the NPC is open

E.g., kick the door in front of the NPC
Behavior Trees (BTs)

- NPC behavior based on more refined conditions and strategies
  - Tasks have a common basic structure: they are given CPU time to do something and return success or failure
  - Leaf tasks: check a condition or execute some code
  - **Composite tasks**: return value depend on child tasks

E.g., succeed if **any** of the child tasks succeed
Behavior Trees (BTs)

- NPC behavior based on more refined conditions and strategies
  - Tasks have a common basic structure: they are given CPU time to do something and return success or failure
  - Leaf tasks: check a condition or execute some code
  - Composite tasks: return value depend on child tasks

E.g., succeed if all of the child tasks succeed
Behavior Trees (BTs)

- NPC behavior based on more refined conditions and strategies
- **Sequence** task and **Selector** task
**Behavior Trees (BTs)**

- NPC behavior based on more refined conditions and strategies

Diagram:
- Door open?
- Move into room

→
NPC behavior based on more refined conditions and strategies

- Door open?
- Move into room
- Move to door
- Move into room
NPC behavior based on more refined conditions and strategies

- Door open?
- Move into room
- Move to door
- Move into room
- Door locked?
- Unlock door
- Kick door
- Door open?
Note that **no search** is involved in this paradigm: the behavior tree is traversed as a kind of **pre-defined program**.
Behavior Trees (BTs)

The way the tree is traversed depends on the implementation, e.g., always start over, keep track of the current node, etc.
Behavior Trees (BTs)

- NPC behavior based on more refined conditions and strategies
- **Non-deterministic** sequence task and selector task
NPC behavior based on more refined conditions and strategies
Behavior Trees (BTs)

- NPC behavior based on more refined conditions and strategies
- Parallel sequence task (similar to sequence)

- E.g., perform move actions while also shooting at target
- Also used to simulate “state-like” behavior by ensuring that a condition holds
Behavior Trees (BTs)

- NPC behavior based on more refined conditions and strategies
- Decorator tasks (wrap objects with same interface)
NPC behavior based on more refined conditions and strategies
Behavior Trees (BTs)

- One of the first commercial video games that used BTs is Halo2 (2004)
- Simple to understand
- Simple to implement
  - ...
- Separation between the work of the programmer and the game designers
- Offers the specification of fine-grained behaviors
Both FSMs and BTs are reactive techniques

The NPC follows a pre-programmed strategy that specifies how the NPC should react in the game depending on the current state/node and conditions that currently hold in the game-world.

A sequence of actions that may be executed in the game, e.g., [move to door, kick door, move into room], need to be represented explicitly in the structure of the FSMs or BTs.
Historically, the vast amount of video games with NPCs use FSMs and BTs for NPC decision making

- Simple to understand/implement
- Separation between programmers and game designers
- Any extensions needed can be handled effectively using programming tricks
- The behavior is strengthened by extra information in the game world that is carefully prepared for NPCs
Reactive Behavior

- A game level from the eyes of an NPC
Reactive Behavior

- A game level from the eyes of an NPC
Reactive Behavior

- The situation today
  - Open worlds with *increasing available interactions*
  - NPCs need to be *autonomous*, with their *own agenda*, goals, personality
Reactive Behavior

- The situation today
The situation today

- Under these circumstances, maintaining the possible and applicable interactions using reactive techniques becomes complex and difficult
- The need for more flexible techniques arises
Reactive Behavior

- The situation today
Goal Oriented Action Planning (GOAP)

- Video Games:
  - Finite State Machines
  - Decision Diagrams
  - Behavior Trees
  - Goal Oriented Action Planning

- Academic AI on agents:
  - Knowledge representation, First-order logic, Classical planning, Planning with preferences, …
  - Belief-Desire-Intention architecture, Agent-based programming, …
  - Probabilistic reasoning, Bayesian networks, Utility theory, Markov Decision Processes, …
Replace the pre-defined strategies with a description of goals and available actions
Replace the pre-defined strategies with a description of goals and available actions.
Replace the pre-defined strategies with a description of goals and **available actions**

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move into room</td>
<td>Door open</td>
<td>In room</td>
</tr>
<tr>
<td>Move to door</td>
<td>-</td>
<td>At door</td>
</tr>
<tr>
<td>Unlock door</td>
<td>Hold key</td>
<td>Door open</td>
</tr>
<tr>
<td>Kick door</td>
<td>-</td>
<td>Door open</td>
</tr>
</tbody>
</table>
Goal Oriented Action Planning (GOAP)

- Replace the pre-defined strategies with a description of **goals** and available actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move into room</td>
<td>Door open</td>
<td>In room</td>
</tr>
<tr>
<td>Move to door</td>
<td></td>
<td>At door</td>
</tr>
<tr>
<td>Unlock door</td>
<td>Hold key</td>
<td>Door open</td>
</tr>
<tr>
<td>Kick door</td>
<td></td>
<td>Door open</td>
</tr>
</tbody>
</table>
Goal Oriented Action Planning (GOAP)

- **Search in real-time for a strategy** that achieves the goal in the current state

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move into room</td>
<td>Door open</td>
<td>In room</td>
</tr>
<tr>
<td>Move to door</td>
<td>-</td>
<td>At door</td>
</tr>
<tr>
<td>Unlock door</td>
<td>Hold key</td>
<td>Door open</td>
</tr>
<tr>
<td>Kick door</td>
<td>-</td>
<td>Door open</td>
</tr>
</tbody>
</table>

Diagram showing the goal state "In room" with moves and effects.
Goal Oriented Action Planning (GOAP)

- Advantages
  - Easy to manage a large number of generated behaviors
  - Able to achieve different behaviors that satisfy the given requirements under different conditions without explicitly listing the resulting strategies

- But it needs to solve planning problems in a few frames!
Behavior Trees (BTs)

- One of the first commercial video games that used BTs is Halo2 (2004)
- Simple to understand
- Simple to implement
  - ...
- Separation between the work of the programmer and the game designers
- Offers the specification of fine-grained behaviors
Goal Oriented Action Planning (GOAP)

- One of the first commercial video games that used GOAP is FEAR (2005)
- Not so simple to understand
- Not so simple to implement
  - ...
- Not so clear separation between the work of the programmer and the game designers
- The specification of fine-grained behaviors is actually tricky
Some details about GOAP in FEAR:

- One AI programmer responsible for NPC behavior

- Idea: Different behaviors can be achieved among characters by using GOAP and providing each character with same goals but a different set of available actions
Goal Oriented Action Planning (GOAP)

- **Soldier**
- **Assassin**
- **Rat**
Goal Oriented Action Planning (GOAP)

- Simplifying STRIPS planning:
  - Literals are stored as variables (essentially having one argument)
  - The state is stored as an array of a fixed size
  - Search goes up to depth …3

- A* for path finding…
- A* also for planning!
Reactive Planning Vs. Classical Planning

- **HALO2 (2004)**
  - Since then BTs have become a standard for NPC behavior

- **FEAR (2005)**
  - Since then GOAP has not picked up much speed
Reactive Planning Vs. Classical Planning

Behavior Trees

Goal Oriented Action Planning
Reactive Planning Vs. Classical Planning

- Behavior Trees
- Goal Oriented Action Planning

- A combination of these techniques?
  - BTs for reactive decision making
  - GOAP for tactical decision making
Amazing tools available for (indie) game developers!
Artificial Intelligence and Video Games

- Source available!
Material

- Sections 5.3, 5.4