Making Simple Decisions
CS3523
AI for Computer Games
The University of Aberdeen

Contents

Motivating Question

- How can we program rules of behaviour for a Non-Player Character (NPC)?
- Constraints:
  - Developers must be able to: create, test, debug, modify easily.
  - Should enhance player experience:
  - Performance (care about speed).
  - Sufficiently interesting and lifelike for players.

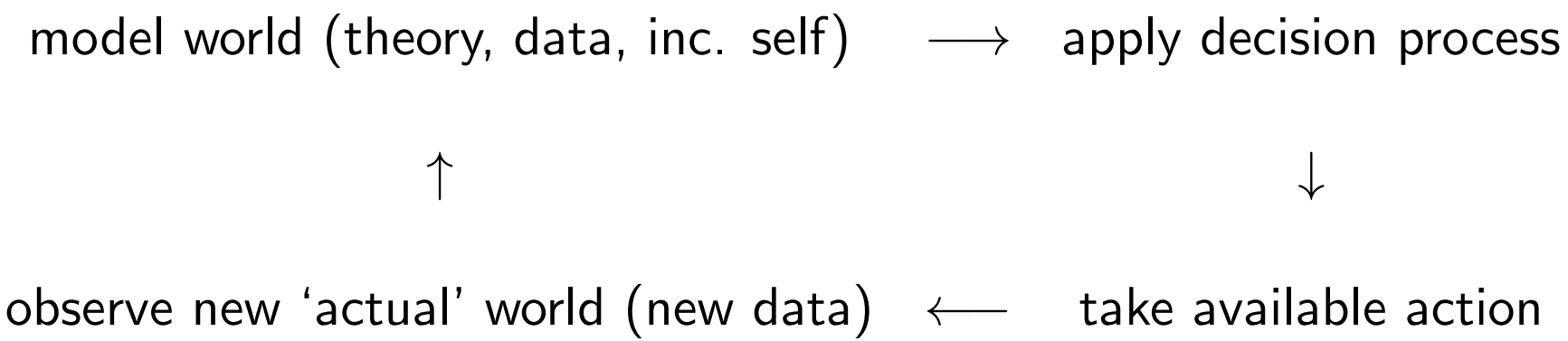
Agent-world interaction

- Think about some simple interaction between an agent and its world.
- Agent wants to influence changing world through action.

$$\text{world (inc. agent)} \xrightarrow{\text{chosen action}} \text{new world}$$

Repeated decision

- Usually, will want to have an agent that continually takes decisions.
- May have a feedback loop like:



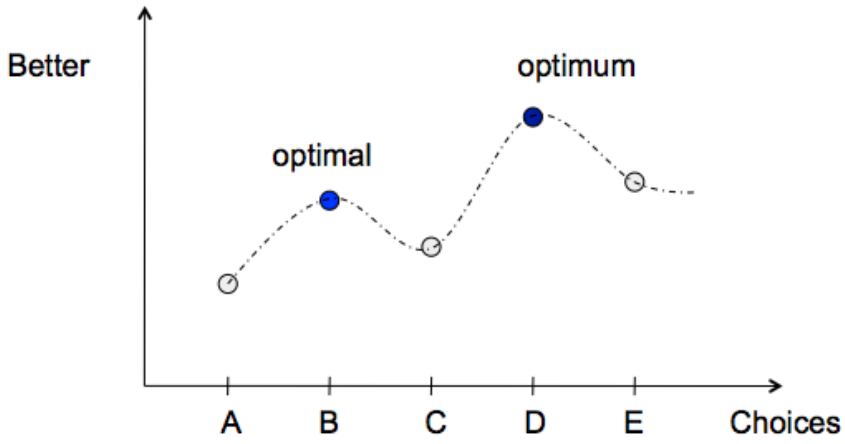
Decision Making

- For a rational agent, decision making leads to action (i.e. behaviour).
- Many subtleties and complexities here. Some are deep/philosophical. How does one define
  - rational
  - agent
  - decision making
  - world?
- Some are more technical
  - Decision making may have a cost; this may have to be factored into the decision!
    - e.g. may decide not to decide, to bury head in the sand.
    - Inaction is an action: choose explicitly to do nothing.
  - Other 'agents' may be choosing and acting simultaneously.
- As a game designer we set the rules, so presumably we are free to pay just as much attention to these issues as we wish, as long as we can satisfy the end-user.
- Our agents may not have to be subject to the rules of the Natural, Social or Behavioural Sciences.

Optimizing Agents

- Consider an agent taking a single decision.
- The agent wants to be select an action that will lead to an outcome that is desirable. So some actions will be better than others.
- Ideally, the decision process should return the optimum (best) action (to give the best behaviour).
- If decision-maker (DM) knows what the outcome of each choice would be, and can optimize over these, then DM has solution.
- This requires DM to know (be coded with, or have access to):
  - what the current state of the world is
  - how the world changes for each choice
  - preferences between these outcomes.

Optimization

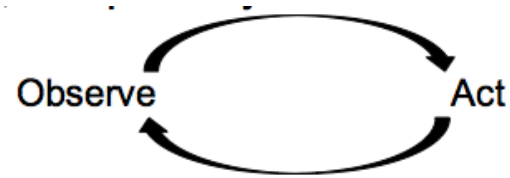


Search

- Search is a major part of AI
  - Search to find out the available actions
  - Search of the world,
    - e.g., for a path, or for treasure on a game map.
- A lot of AI involves optimization.
  - A lot of optimization is about search for the optimum solution, or,
  - search for an optimal (and good-enough) solution; this is known as satisficing.
- You'll spend some time looking at search, particularly pathfinding, starting from next week.

Simple Reflexes

- The decision-making that we will look at today is that done by simple-reflex agents.
- They observe the world (state) and then react in fixed, 'simple' ways.
- They usually do this kind of thing continually, e.g. at every clock-tick of the game.



- They don't search for sophisticated choices.
- They don't build a sophisticated model.

Decision Trees (1)

- Decision trees used to compute (reflex) choices in state of world.
- Agent can find out properties of world state, and use this information to choose its reaction.
- Use a simple kind of computable function that maps world-states to simple actions, for any given decision:

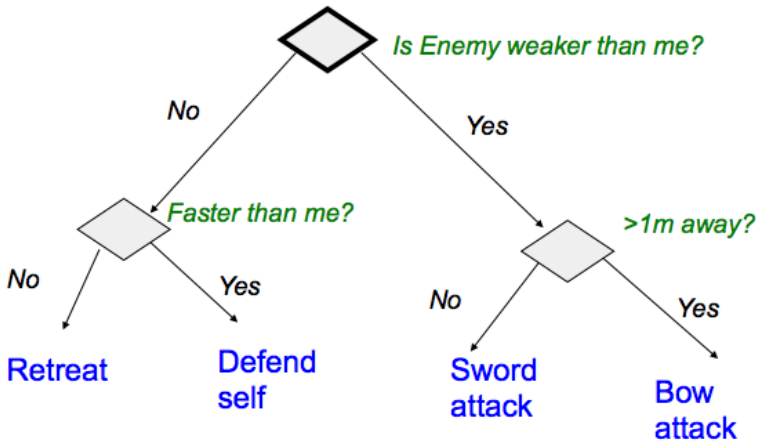
$$\text{DecTree} : \text{States} \longrightarrow \text{SimpleActions} \ .$$

- So, if it is given a state s, then an action DecTree(s) is chosen.

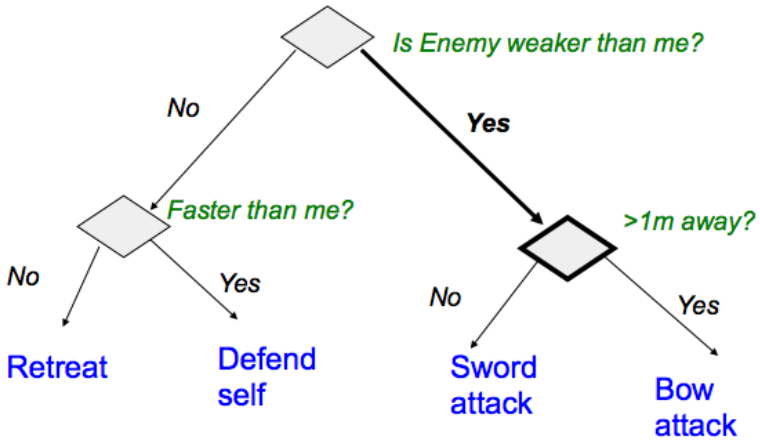
Decision Trees (2)

- Can be coded using:
  - conditional statements (`if-then`), or
  - rules from a dedicated rule-language.
- Pragmatics of this choice in games
  - fast
  - easy to understand
  - See Millington for more details.

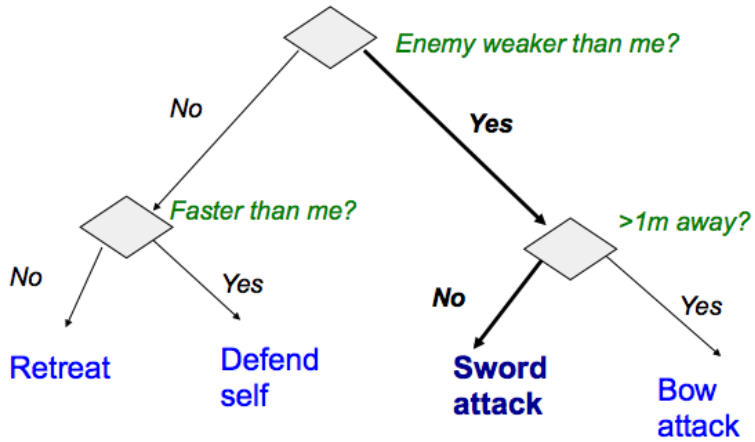
E.G.: World with enemy seen



E.G.: enemy observed to be weak



E.G.: enemy observed to be weak and close



Decision Tree Issues

- Not required to be binary: nodes can have many children.
- Performance:
  - choose order of interrogation nodes.
- Generalizations:
  - Edges allowed to merge at a lower node: decision graphs (but usually must be directed acyclic)
  - Randomness: (slides below).
- Different decision trees for different NPCs, or for the same NPC in different modes (also below)
  - E.g. separate fight/flee-mode, from attack-mode.

Decision and Learning

- We have focussed on a version of decision trees that simply calculates choice (leading to behaviour) by a sequence of interrogations of states of the world, with no optimization.
- Which is the best action can be learned (won't discuss today).

More Complicated Decision Trees

- Different type of 'decision-tree' used for economic, managerial (and some AI) decisions. There are various different forms of these trees in the literature.
- Used for more-sophisticated choices (not just reflexes). Require optimization procedures.

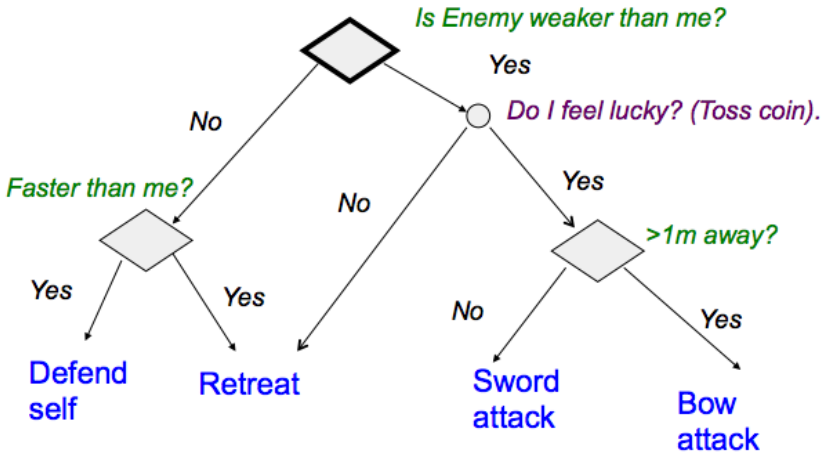
Randomness

- Randomness is very important.
- World changes too hard to predict exactly.
- NPC behaviour might be random.
- Human player might be too hard to predict.
- Agents may not be able to predict each others behaviour.
- Random algorithms important computational tools, when dealing with large state and search spaces.

Random NPC

- Random behaviour of NPC may be best.
- Easy to add a node to decision tree which tests a random number.

EG: Decision graph with random node



Random NPC — Sampling Issue

- If game 'ticks' every 20ms, do we really need and want the random values to be re-sampled, and the NPCs to change their behaviour this often?
  - e.g. random attack/flee choice
  - e.g. goal location
  - Should the NPC change its behaviour every 20ms?
  - Probably not.
- Be careful about how often behaviour changes.

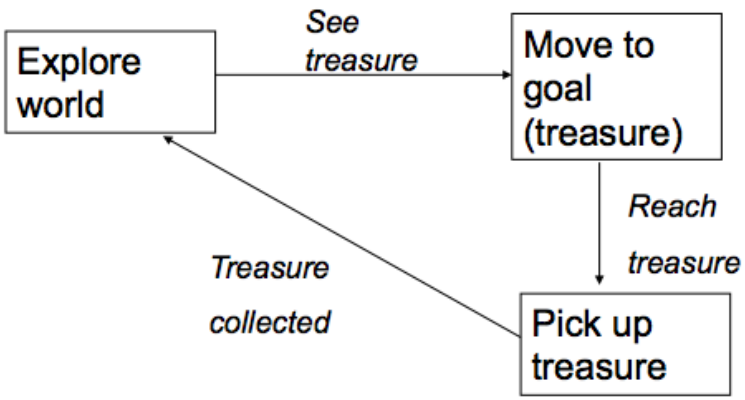
More Persistent NPC State

- Better to have NPC in a definite **state**, which is only changed at sensible intervals.
  - E.g. state: fight, flee
- Behaviour determined by one decision tree, which interrogates the state of *both* the NPC *and* the world, and which runs every game tick. (e.g. 20ms).
- NPC state changed by a *separate* decision tree
  - e.g. when environ. changes, e.g. enemy gone, or
  - e.g. after a fixed period, say 10 mins.
- OO-approach really helps here.

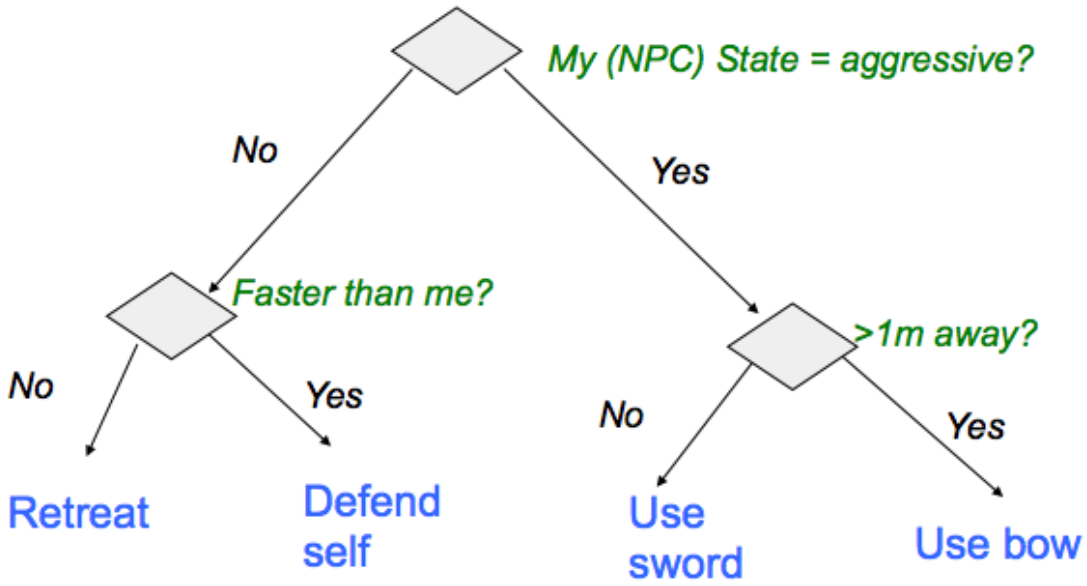
State Machines

- Alternative way to do decisions
- Focuses on state
- Most common today in games
  - According to Millington
- a.k.a FSM (Finite State Machine), Finite State Automata

Example: Find treasure FSM



Example: Use of NPC states in decision tree



Quake code: set state (a transition)

```
if (enemy_range == RANGE_MELEE) {

  // Don't always melee in easy mode (if skill points to value == 0)
  if (skill->value == 0 && (rand()&3) ) return false;

  // If it is a monster which melees, set attack state to AS_MELEE
  if (self->monsterinfo.melee)
     self->monsterinfo.attack_state = AS_MELEE;

  else                          // else set attack state to AS_MISSILE
    self->monsterinfo.attack_state = AS_MISSILE;

  return true;
}
```

Quake code: use of state in decision

```
// In AS_MISSILE state attack with missile
    if (self->monsterinfo.attack_state == AS_MISSILE) {
            ai_run_missile (self);
            return true;
    }

// In AS_MELEE state attack in melee
    if (self->monsterinfo.attack_state == AS_MELEE){
            ai_run_melee (self);
            return true;
    }
```

FSM with Decision Trees

- Most of quake code is about special case handling, not "being smart"
  - e.g. monster which can't melee
  - e.g. easy mode
- Similar to other game software
  - 80% (90%) is special case handling
- Brittleness in unusual situations is major problem with AI in general.

Hierarchical State Machines

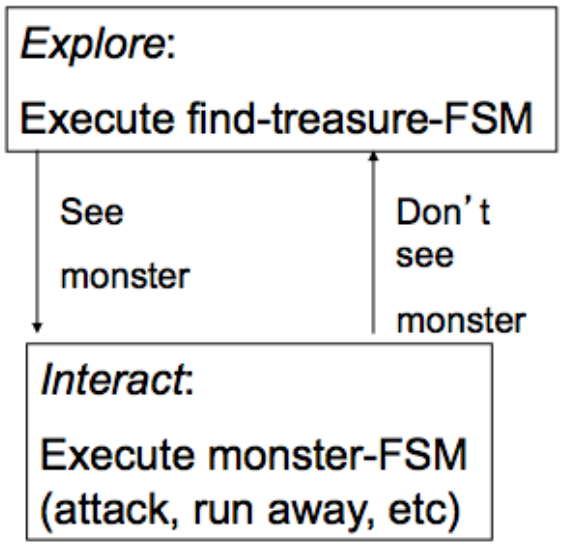
- Usually have:
  - many states in games
  - no direct transition between most states
  - certain, similar transitions occur repeatedly.
- State is often naturally hierarchical.
- Can ‘chunk’ state machines and combine into hierarchical state machines.
- Quite natural from a coding point of view.

HSM Alarm

E.G., used for alarms:

- Can have one FSM for basic background behaviour, and one (alarm) for behaviour in presence of enemy.
- When enemy seen:
  - interrupt background-FSM with alarm-FSM to deal with enemy
- Revert to background-FSM when enemy gone.

HSM Alarm Example



State machines elsewhere

State machines very common in CS as a whole:

- Compilers
- Software engineering (UML)
- Hardware design
- Theory of computing, logic
- Linguistics

Example: dog

- What states does a dog have
  - Quake, Sims
  - real life?
- What drives state transition?

How good are these?

- In both of the above approaches (FSM and dec. tree) the choice of behaviour is made according to some (simple or complicated) rules, given the world.
- R & N call this a *reflex agent* (Section 2.4).
- How close to the optimal (or even an optimum solution) are such decisions:
  - For the player of the game?
  - For the designer?

Simple AI Works in Complex Games

- Simple state machines and decision trees can give rise to *intelligent-looking* behaviour in a complex world.
- General finding in AI that apparent intelligence can come from:
  - complex algorithms in simple world, or
  - simple algorithms in complex world.

Problem

- User can find a bug or poorly handled special case, and exploit this
  - e.g. monster uses weak sword instead of mega-powerful bow if in sword range.
- Deeper AI would help:
  - Want common sense NPC behaviour
  - Such behaviour a long-term goal of AI.
  - Could be expensive to build, test, verify.

Game Engines

- Lots of packages developed by individuals for:
  - Decision trees,
  - FSM,
  - HSM.
- Are there now any standard and widely-used open-source engines with these?

Reading

You might be interested to read more in:

- Millington, Chapter 5.
- Russell and Norvig, 2nd ed., bits of chapters 2 and 16.
- This is not required.

Attendance

- Attendance at practicals is mandatory.
- You will not be allowed to sit the exam if you fail to attend (without good cause).
- Most of the students who fail this course are the ones who don't show up.
- You will find later practicals very hard if you don't do the earlier ones first.
- No student has performed well who has regularly skipped this class.