COMP250: Artificial Intelligence

3: Game Theory and Planning

Game theory

Game theory

- A branch of mathematics studying **decision making**
- A **game** is a system where one or more **players** choose **actions**; the combination of these choices lead to each agent receiving a **payoff**
- Important applications in economics, ecology and social sciences as well as AI

The ~~Prisoner's~~ Student's Dilemma

- Two students, **Alice** and **Bob**, are suspected of copying from each other
- Each is offered a deal in exchange for information
- Each can choose to **betray** the other or stay **silent** — but they **cannot communicate** before deciding what to do
- If **both stay silent**, both receive a C grade
- If **Alice betrays Bob**, she receives an A whilst he gets expelled
- If **Bob betrays Alice**, he receives an A whilst she gets expelled
- If **both betray each other**, both get an F

Payoff matrix

	A silent	A betray
B silent	A: 50 B: 50	A: 70 B: -100
B betray	A: -100 B: 70	A: 0 B: 0

Nash equilibrium

- Consider the situation where both have chosen to betray
- Neither person has anything to gain by switching to silence, assuming the other person doesn't also switch
- Such a situation is called a **Nash equilibrium**
- If all players are **rational** (in the sense of wanting to maximising payoff), they should converge upon a Nash equilibrium

Does every game have a Nash equilibrium?

	A rock	A paper	A scissors
B rock	A: 0 B: 0	A: +1 B: -1	A: -1 B: +1
B paper	A: -1 B: +1	A: 0 B: 0	A: +1 B: -1
B scissors	A: +1 B: -1	A: -1 B: +1	A: 0 B: 0

Nash equilibrium for Rock-Paper-Scissors

- Committing to any choice of action can be **exploited**
- E.g. if you always choose paper, I choose scissors
- If we try to reason naïvely, we get stuck in a loop
  - If I choose paper, you'll choose scissors, so I should choose rock, but then you'll choose paper, so I'll choose scissors, so you'll choose rock, so I choose paper...
- The optimum strategy is to be **unpredictable**
- Choose rock with probability $\frac{1}{3}$, paper with probability $\frac{1}{3}$, scissors with probability $\frac{1}{3}$

Mixed strategies

- A **mixed strategy** assigns probabilities to actions and chooses one at random
- In contrast to a **pure** or **deterministic strategy**, which always chooses the same action
- If we allow mixed strategies, **every game has at least one Nash equilibrium**

Guess $\frac{2}{3}$ of the average

- Everyone guesses a real number (decimals are allowed) between 0 and 100 inclusive
- The winner is the person who guesses closest to $\frac{2}{3}$ of the mean of all guesses
- Example:
  - If the guesses are 30, 40 and 80...
  - ... then the mean is $\frac{30+40+80}{3} = 50$...
  - ... so the winning guess is 30, as this is closest to $\frac{2}{3} \times 50 = 33.333$

The rational guess

- The average can't possibly be greater than 100
- So no rational player would guess a number greater than 66.666
- Which means the average can't possibly be greater than 66.666
- So no rational player would guess greater than 44.444
- Which means the average can't possibly be greater than 44.444
- So no rational player would guess greater than 29.629
- ... and so on ad infinitum
- So the only **rational** guess is 0, as every rational player should guess 0 and $\frac{2}{3}$ of 0 is 0

Rationality

- Rationality is a useful assumption for mathematics and AI programmers
- However it's important to remember that **humans aren't always rational**
- E.g. in guess $\frac{2}{3}$, if enough players deviate from the rational guess of 0, then 0 is no longer the best guess!

Planning

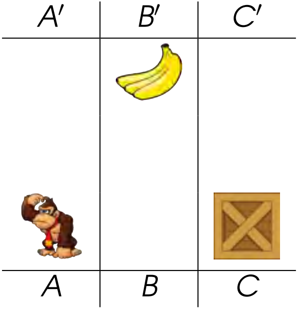
Planning

- An **agent** in an **environment**
- The environment has a **state**
- The agent can perform **actions** to change the state
- The agent wants to change the state so as to achieve a **goal**
- Problem: find a sequence of actions that leads to the goal

STRIPS planning

- **St**anford **R**esearch **I**nstitute **P**roblem **S**olver
- Describes the state of the environment by a set of **predicates** which are true
- Models a problem as:
  - The **initial state** (a set of predicates which are true)
  - The **goal state** (a set of predicates, specifying whether each should be true or false)
  - The set of **actions**, each specifying:
    - Preconditions (a set of predicates which must be satisfied for this action to be possible)
    - Postconditions (specifying what predicates are made true or false by this action)

STRIPS example



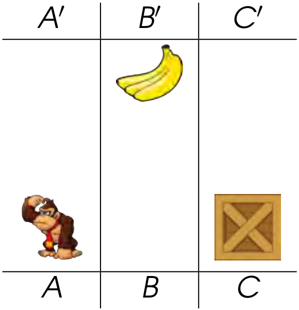
Initial state:

```
At(A),
BoxAt(C),
BananasAt(B')
```

Goal:

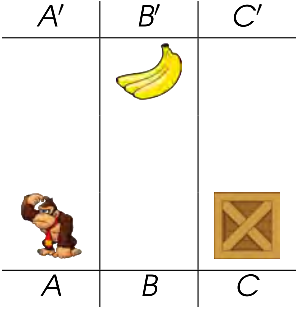
```
HasBananas
```

STRIPS example — Actions




```
Move(x, y)
  Pre:  At(x)
  Post: !At(x), At(y)

ClimbUp(x)
  Pre:  At(x), BoxAt(x)
  Post: !At(x), At(x’)

ClimbDown(x’)
  Pre:  At(x’), BoxAt(x)
  Post: !At(x’), At(x)

PushBox(x, y)
  Pre:  At(x), BoxAt(x)
  Post: !At(x), At(y),
        !BoxAt(x), BoxAt(y)

TakeBananas(x)
  Pre:  At(x), BananasAt(x)
  Post: !BananasAt(x), HasBananas
```

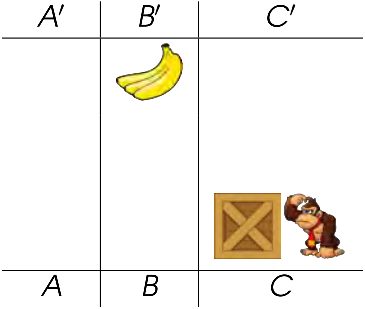
STRIPS example — Solution



State:

```
At(A),
BoxAt(C),
BananasAt(B')
```

STRIPS example — Solution



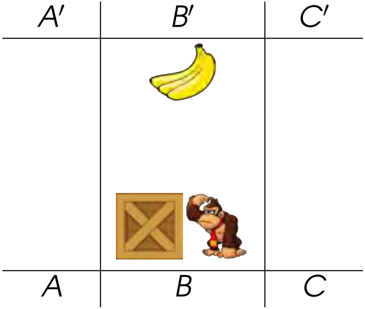
Action:

```
Move(A, C)
```

State:

```
At(C),
BoxAt(C),
BananasAt(B')
```

STRIPS example — Solution



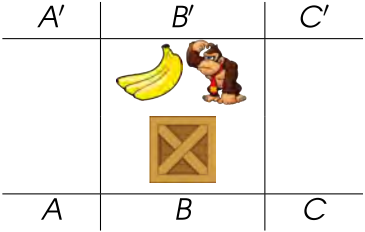
Action:

```
PushBox(C, B)
```

State:

```
At(B),
BoxAt(B),
BananasAt(B')
```

STRIPS example — Solution



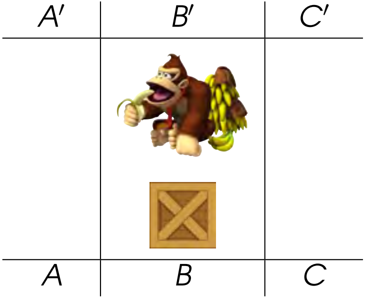
Action:

```
ClimbUp(B)
```

State:

```
At(B’),
BoxAt(B),
BananasAt(B’)
```

STRIPS example — Solution



Action:

```
TakeBananas(B')
```

State:

```
At(B'),
BoxAt(B),
HasBananas
```

Finding the solution

- For a given state, we can construct a list of all **valid actions** based on their **preconditions**
- We can also find the **next state** resulting from each action based on their **postconditions**
- We can construct a **state-action graph**
  - Nodes: environment states
  - Edges: actions
- We can then **search** this graph to find the optimum path from the current state to a goal state

Searching for the solution

- We have a **tree**, which is a type of **graph**
- We have an **initial node** within this tree
- Want to find a **sequence of edges** that leads to a **goal node**
- Does this sound familiar?
- Very similar to **pathfinding**, so can use the same algorithms (recall from COMP280 session 8)
  - Depth-first search
  - Breadth-first search
  - Dijkstra's algorithm (same as BFS unless actions have non-uniform **cost**)
  - A* (if we have a suitable **heuristic**)

GOAP

GOAP

- **G**oal **O**riented **A**ction **P**lanning
- Originally developed for F.E.A.R. (2005), since used in several games
- A modified version of STRIPS specifically for real-time planning in video games

GOAP

- Each agent has a **goal set**
  - Multiple goals with differing **priority**
  - Goals are like in STRIPS — sets of predicates that the agent wants to satisfy
- Each agent also has a set of **actions**
  - Like in STRIPS — actions have preconditions and postconditions
  - Unlike STRIPS, each action also has a **cost**

Action sets

- Different types of agent could have the **same goals** but **different action sets**
- This will result in those agents achieving those goals in **different ways**
- NB this doesn't have to be explicitly coded — it **emerges** from the GOAP system
- E.g. this was used by the F.E.A.R. team to quickly add new enemy types

Action sets

Soldier

Action

1. AI/Actions/Attack
2. AI/Actions/AttackCrouch
3. AI/Actions/SuppressionFire
4. AI/Actions/SuppressionFireFromCover
5. AI/Actions/FlushOutWithGrenade
6. AI/Actions/AttackFromCover
7. AI/Actions/BlindFireFromCover
8. AI/Actions/AttackGrenadeFromCover
9. AI/Actions/AttackFromView
10. AI/Actions/DrawWeapon
11. AI/Actions/HolsterWeapon
12. AI/Actions/ReloadCrouch
13. AI/Actions/ReloadCovered
14. AI/Actions/InspectDisturbance
15. AI/Actions/LookAtDisturbance
16. AI/Actions/SurveyArea
17. AI/Actions/DodgeRoll
18. AI/Actions/DodgeShuffle
19. AI/Actions/DodgeCovered
20. AI/Actions/Uncover
21. AI/Actions/AttackMelee

Assassin

Action

1. AI/Actions/Attack
2. AI/Actions/InspectDisturbance
3. AI/Actions/LookAtDisturbance
4. AI/Actions/SurveyArea
5. AI/Actions/AttackMeleeUncloaked
6. AI/Actions/TraverseBlockedDoor
7. AI/Actions/UseSmartObjectNodeMounted
8. AI/Actions/MountNodeUncloaked
9. AI/Actions/DismountNodeUncloaked
10. AI/Actions/TraverseLinkUncloaked
11. AI/Actions/AttackFromAmbush
12. AI/Actions/DodgeRollParanoid
13. AI/Actions/AttackLungeUncloaked
14. AI/Actions/LopeToTargetUncloaked

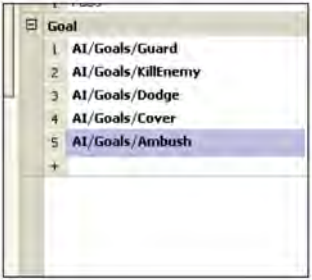
Rat

Action

1. AI/Actions/Animate
2. AI/Actions/Idle
3. AI/Actions/GotoNode
4. AI/Actions/UseSmartObjectNode

Layering

- Goal set allows different behaviours with different priorities to be **layered**
- E.g. enemy AI in F.E.A.R.:



Implementing GOAP

- An **abstracted** view of the game world is used for planning
- Represented as a fixed-length array (or struct) of values
- Predicates (preconditions, postconditions, goals) represented in terms of this array representation
- Most implementations also allow for **programmatic** preconditions (e.g. calling the pathfinding system to check availability of a path)

Implementing GOAP

- Not difficult to implement
- Open-source implementations do exist
- Not built into Unity or Unreal, but asset store packages are available

Finding the plan

- As in STRIPS, we can build a **tree** whose nodes are world states and edges are available actions
- Since actions have costs, we can use Dijkstra or A* to find the lowest cost path to the goal
- Plan is a **queue** of actions that the agent then executes
- If the plan is interrupted or fails then the agent can **replan**

GOAP vs behaviour trees

- BT: Designer specifies "how"
- GOAP: Designer specifies "what" — "how" is in whatever system is used to implement actions (FSMs in F.E.A.R.; could use BTs or hand coding)
- Both: actions (tasks in BT) are modular and reusable between agents
- GOAP: goals are also modular and reusable
- BT: goals are not represented explicitly
- BT can be classified as **authored behaviour**
- GOAP can be classified as **computational intelligence**