Алгоритмы принятия решений
Decision making

Ст. преп. каф. ПМП Пучкин Максим Валентинович

Определения

- Decision making – the ability of a character to decide what to do. Given a set of knowledge, we need to generate a corresponding action from a set of possible actions.
- Методы принятия решений для игровых агентов, как правило, на основе некоторой входной информации (input);
- Под решением понимаем выбор одного элемента из некоторого конечного (или бесконечного?) множества – множества стратегий, множества ходов, множества состояний и проч.
- Таким образом, формализуем:

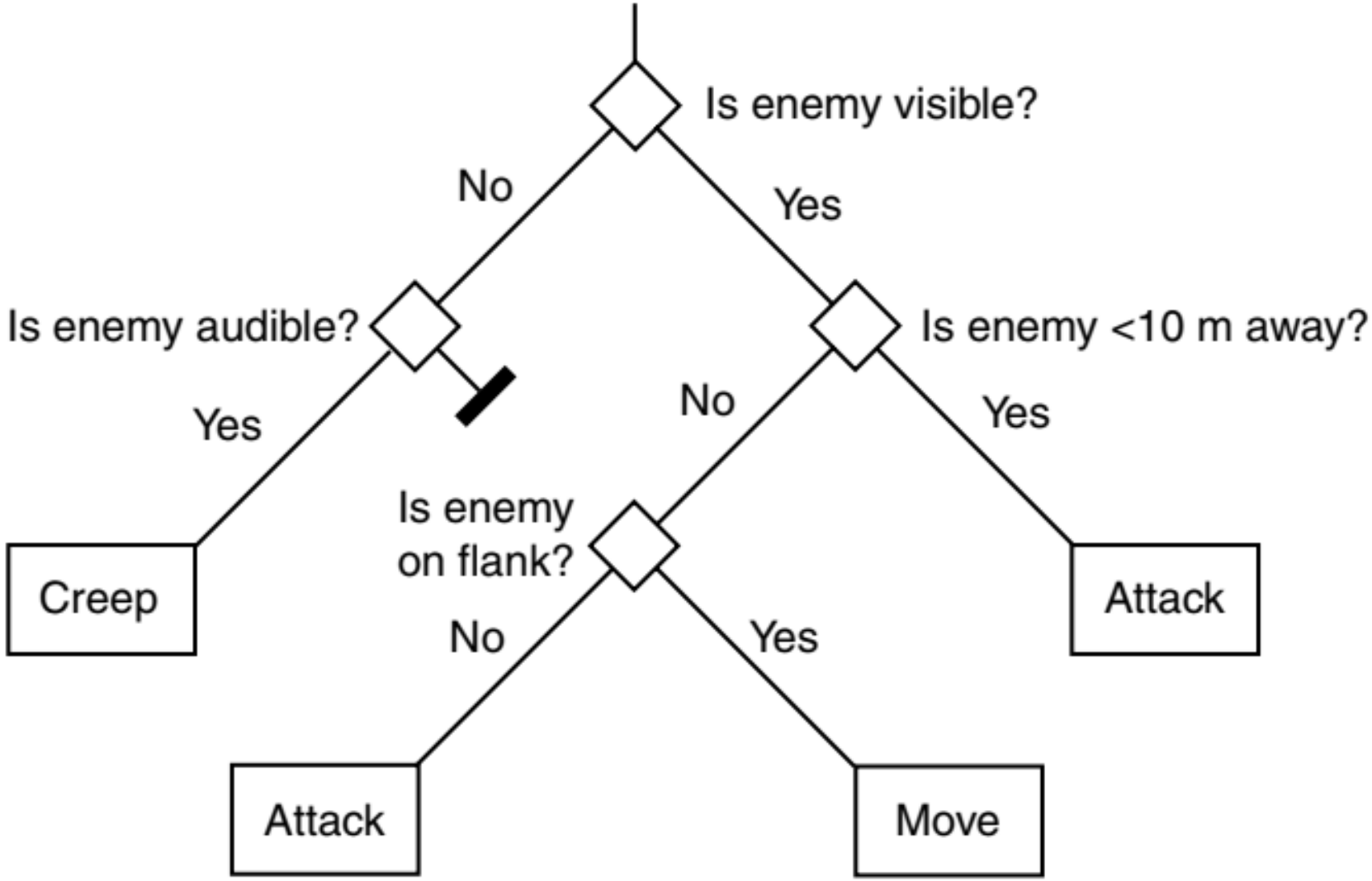
$$Decision: \ States \times Input \rightarrow Reactions$$

- *States* – множество возможных состояний агента, к которым можно отнести и управляющие воздействия от планировщика верхнего уровня.
- *Reactions* – множество возможных реакций агента.

Подходы

- Много *if … then* – а почему бы нет?
- Decision tree – деревья принятия решений. Довольно легко строятся, могут модифицироваться в процессе игры. Высокая эффективность;
- Finite state machine (FSM) – конечные автоматы. Эффективны, легко реализуемы и модифицируемы, соответствую человеческой логике.
- Expert system (Rule-based system) – «тяжёлая артиллерия». Используются редко – нужны «движки», навыки, а отличия от FSM на простых задачах невелики. Умеют рассуждать разумно.
- Artificial neural networks – сложно, непонятно, и долго считают.

Decision trees



Decision trees

- Дерево с начальным узлом – root.
- Бинарные – в каждом узле да/нет, но в общем случае могут быть и ветвящимися.
- Листья – отдельный тип, отвечающий за конкретное решение.
- Формально могут быть не деревьями – вопрос оптимизации, впрочем, принципиально реализация не отличается.
- Типы вершин – логические, перечислимые, диапазоны (ООП нам в помощь).
- Знания, как правило, ссылаются на глобальную область, а вот собственно деревья для каждого агента свои.
- Недостаток – начинаем каждый раз от корня (ну, не такой уж и недостаток).

Как построить?

- Ручками (или карандашами) – обычно строятся вручную, благо само представление в виде дерева позволяет уменьшить число ошибок.
- Можно строить автоматически – например, как в игре «Животные», однако возникает вопрос устойчивости к ошибкам.
- Вопросы эффективности – балансировка. Дерево должно быть сбалансированным, но можно ли его балансировать? Подумать!
- При построении используется различные виды вершин, что, строго говоря, добавляет сложностей – для диапазонных типов, к примеру, нужно вносить разбиения.
- Сложность определения релевантных признаков.

Пример

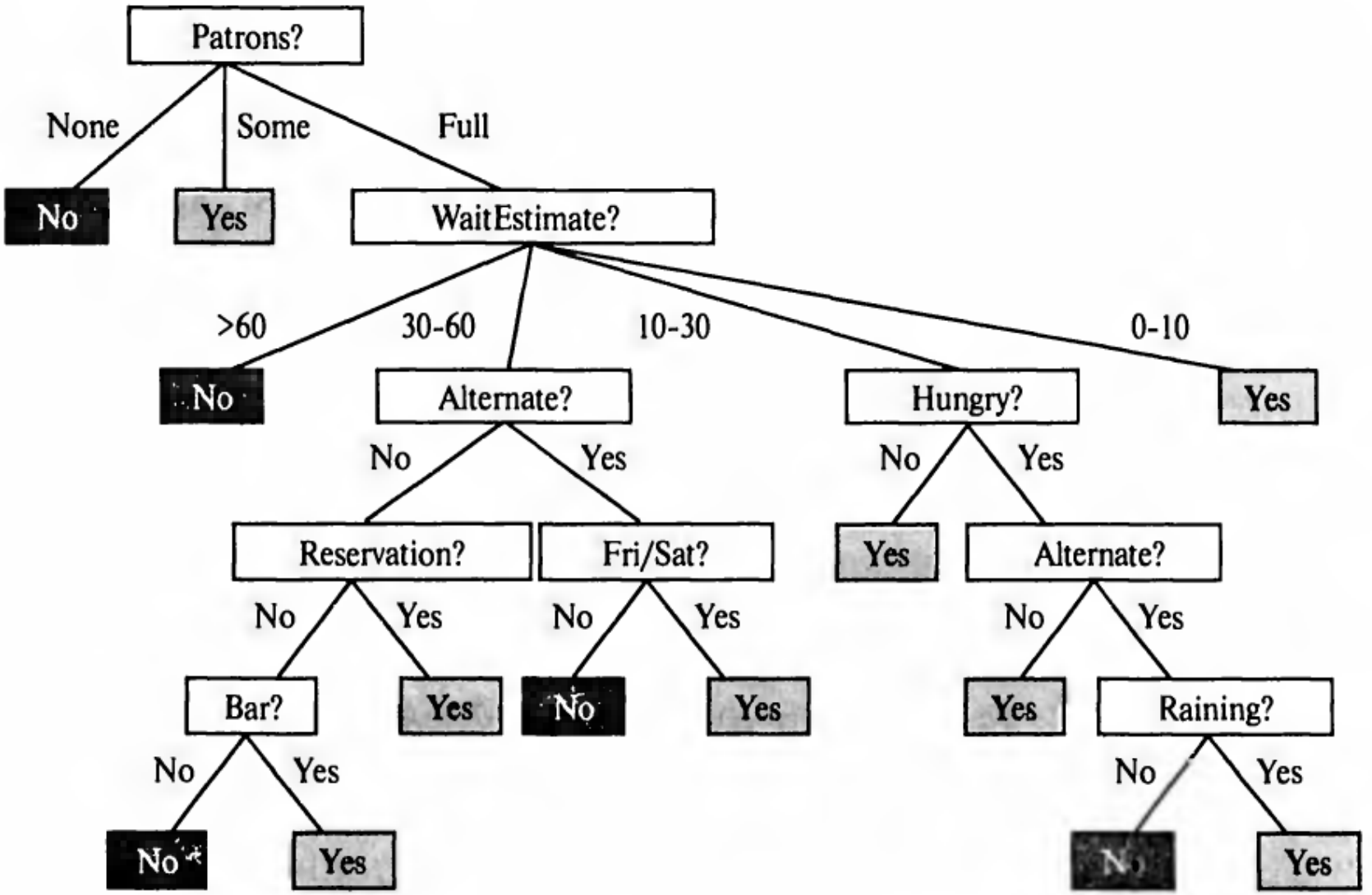


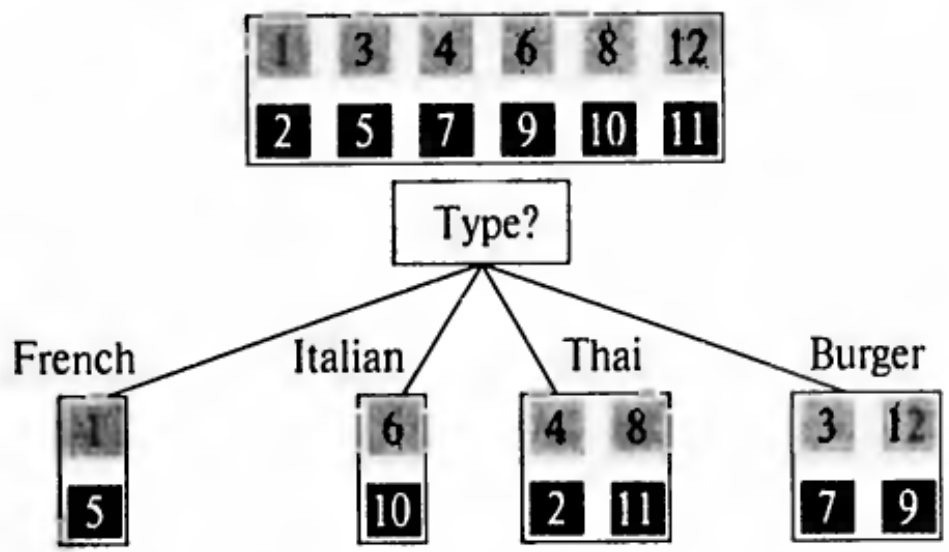
Рис. 18.2. Дерево решений для определения того, следует ли подождать, пока освободится столик

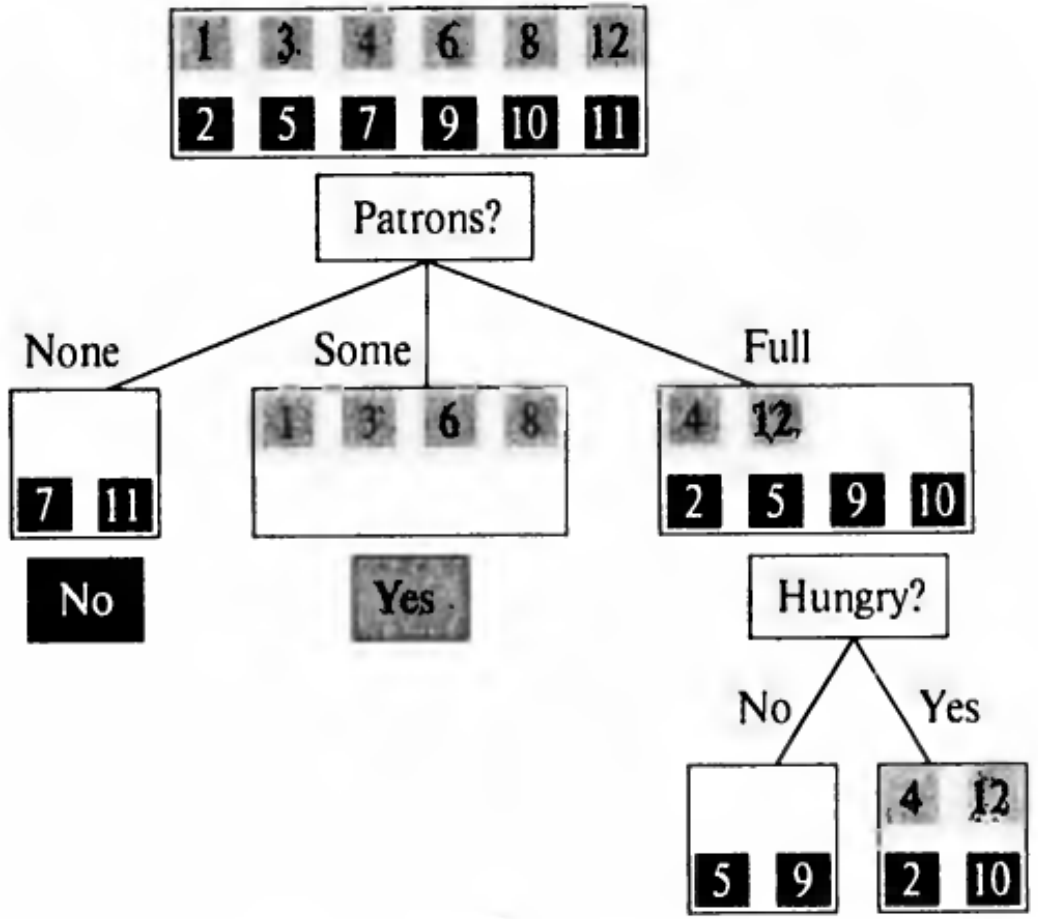
Стюарт Рассел, Питер Норвиг. Искусственный интеллект – современный подход, 2 изд. Стр. 871

Обучающая выборка

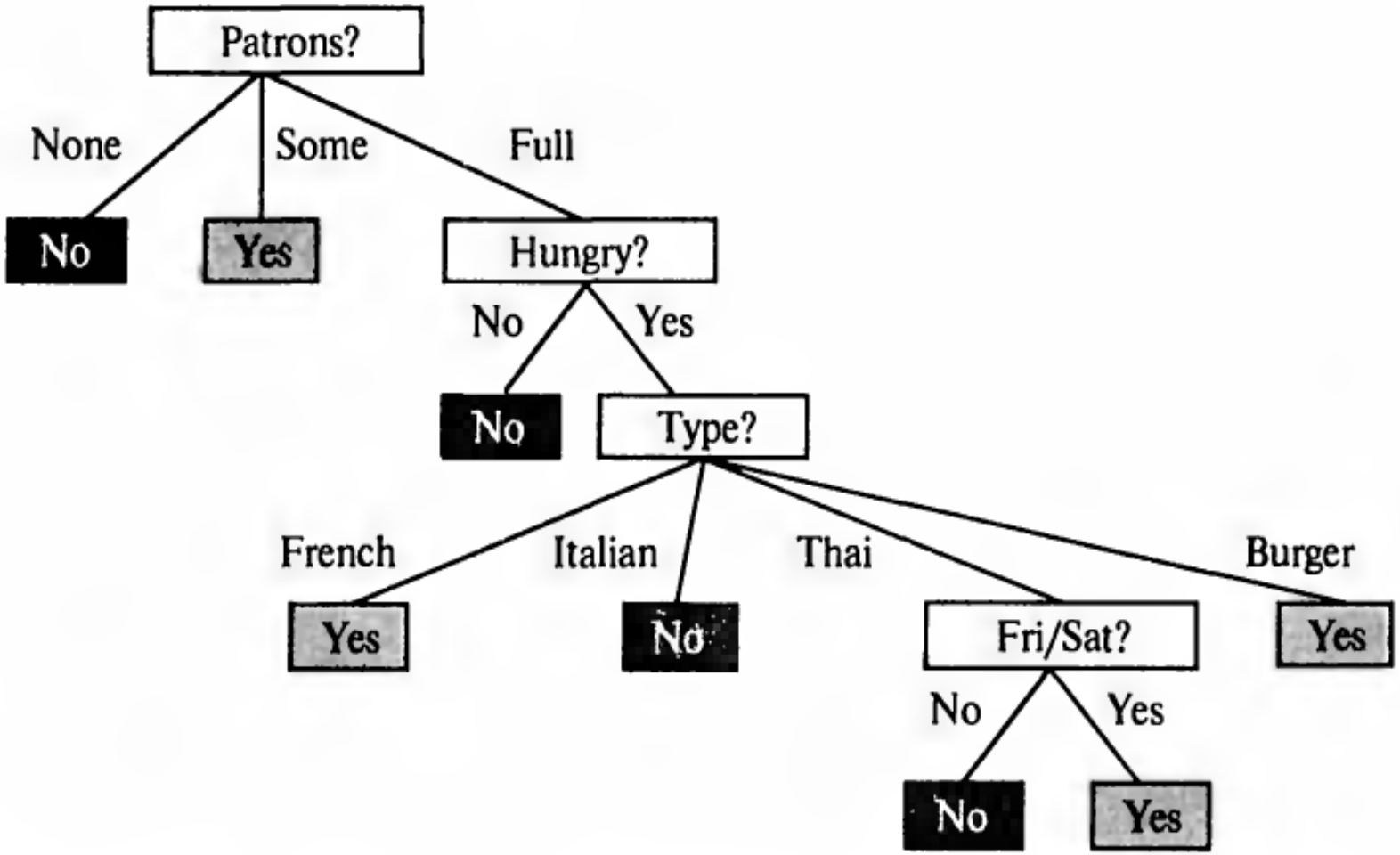
	Атрибуты										Цель
Пример	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	WillWait
X_1	Yes	No	No	Yes	Some	$	No	Yes	French	0-10	Yes
X_2	Yes	No	No	Yes	Full	$	No	No	Thai	30-60	No
X_3	No	Yes	No	No	Some	$	No	No	Burger	0-10	Yes
X_4	Yes	No	Yes	Yes	Full	$	Yes	No	Thai	10-30	Yes
X_5	Yes	No	Yes	No	Full	$	No	Yes	French	760	No
X_6	No	Yes	No	Yes	Some	$	Yes	Yes	Italian	0-10	Yes
X_7	No	Yes	No	No	None	$	Yes	No	Burger	0-10	No
X_8	No	No	No	Yes	Some	$	Yes	Yes	Thai	0-10	Yes
X_9	No	Yes	Yes	No	Full	$	Yes	No	Burger	760	No
X_{10}	Yes	Yes	Yes	Yes	Full	$	No	Yes	Italian	10-30	No
X_{11}	No	No	No	No	None	$	No	No	Thai	0-10	No
X_{12}	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30-60	Yes

Анализ

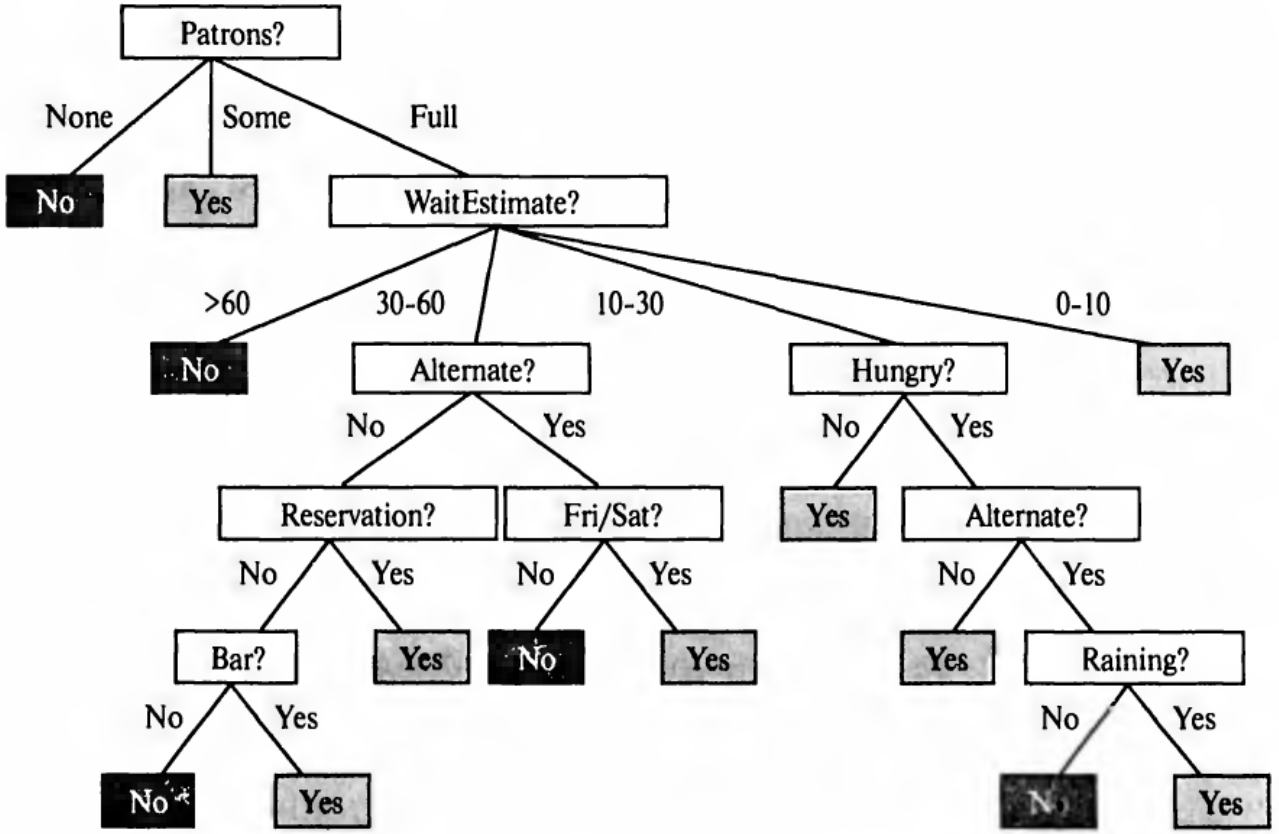

1 3 4 6 8 12
2 5 7 9 10 11
Type?
French
Italian
Thai
Burger
1
5
6
10
4 8
2 11
3 12
7 9

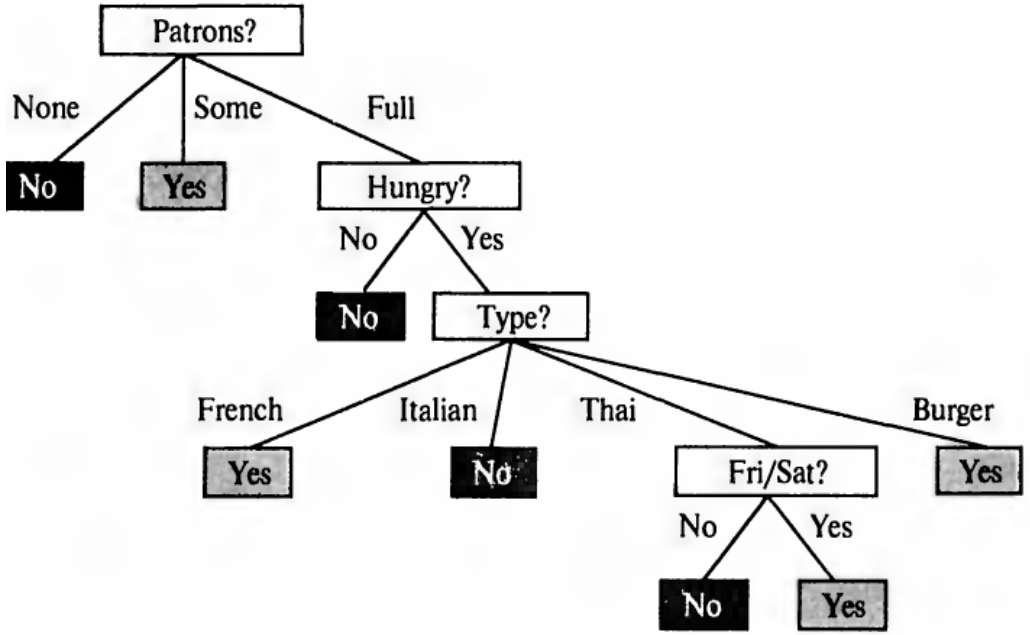

1 3 4 6 8 12
2 5 7 9 10 11
Patrons?
None
Some
Full
7 11
No
1 3 6 8
Yes
4 12
2 5 9 10
Hungry?
No
Yes
5 9
4 12
2 10

Решение


Patrons?
None
Some
Full
No
Yes
Hungry?
No
Yes
No
Type?
French
Italian
Thai
Burger
Yes
No
Fri/Sat?
Yes
No
Yes
No
Yes

Сравнение


Patrons?
None
Some
Full
No
Yes
WaitEstimate?
>60
30-60
10-30
0-10
No
Alternate?
Hungry?
Yes
No
Yes
No
Yes
Reservation?
Fri/Sat?
Yes
Alternate?
No
Yes
No
Yes
No
Yes
Bar?
Yes
No
Yes
Yes
Raining?
No
Yes
No
Yes
No
Yes


Patrons?
None
Some
Full
No
Yes
Hungry?
No
Yes
No
Type?
French
Italian
Thai
Burger
Yes
No
Fri/Sat?
Yes
No
Yes
No
Yes

Выбор признаков

Для бинарных – на основе количества информации (теория Шеннона и Уивера).

$$I\big(P(v_1), P(v_2), \dots, P(v_n)\big) = \sum_{i=1}^{n} -P(v_i)\log_2 P(v_i) \text{ бит}$$

$$I\left(\frac{p}{n+p}, \frac{n}{n+p}\right) = -\frac{p}{n+p}\log_2\frac{p}{n+p} - \frac{n}{n+p}\log_2\frac{n}{n+p}$$

Если есть некоторый атрибут *A*, имеющий *v* различных значений, и делящий обучающее множество *E* на $E_1..E_v$. Каждое подмножество E_i включает p_i положительных примеров, и n_i отрицательных.

Выбор признаков

Остаток информации после перехода по *i*-му атрибуту:

$$Remainder(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

Приращение информации после анализа *i*-го атрибута

$$Gain(A) = I\left(\frac{p}{n + p}, \frac{n}{n + p}\right) - Remainder(A)$$

Таким образом, изначально выбираем тот атрибут, который обеспечивает наибольшее приращение информации при анализе. Далее – аналогично.

Проблемы

Компрометация – классическая проблема обучения, когда алгоритм учится не проблеме, а обучающей выборке.

Overfitting – переобучение. В принципе, верно и для деревьев решений, но выражается несколько по-другому.

Нерелевантные признаки – например, броски монетки в зависимости от дня недели.

Требуется способ проверки значимости атрибутов, и отбрасывание нерелевантных даже в том случае, если они могут влиять на классификацию. А для этого необходимо оценивать значимость атрибутов – например, по тем же самым значениям приращения информации.

Finite State Machines

Finite State Machine – абстракция, имеющая один вход, и, формально, один выход.

Чаще всего рассматривают работу на строках: на вход автомату подаются символы, на выходе формируется отклик. В простейшем случае выход – допускает (accepts) или отвергает (rejects) автомат строку.

Приоритетная область исследования – парсинг строк, синтаксические и лексические анализаторы …, область исследования – теория конечных автоматов.

Автомат в каждый момент времени находится в одном состоянии из множества возможных, и это множество конечно.

Детерминированы – каждое переход из одного состояния в другое однозначно определяется текущим состоянием и информацией на входе автомата.

Определение FSM

Формальное определение:

$$A = (Q, q_0 \in Q, A \in Q, \Sigma, \delta)$$

Q – (states), множество состояний (узлы графа);

q_0 – (initial state), начальное состояние;

A – (accepting state), допускающее состояние;

Σ – (alphabet), входной алфавит;

$\delta: Q \times \Sigma \to Q$ – функция переходов. Может быть определена не на всех элементах множества $\{Q \times \Sigma\}$, в таком случае подразумеваем, что автомат останавливается (неполная функция переходов).

Конечность автомата – два условия, нет самопроизвольных переходов между состояниями, и функция переходов не подразумевает случайность.

Представление и оптимизация FSM

Представление может быть задано таблицами, диаграммами переходов, UML или графами (что-то тут не то!).

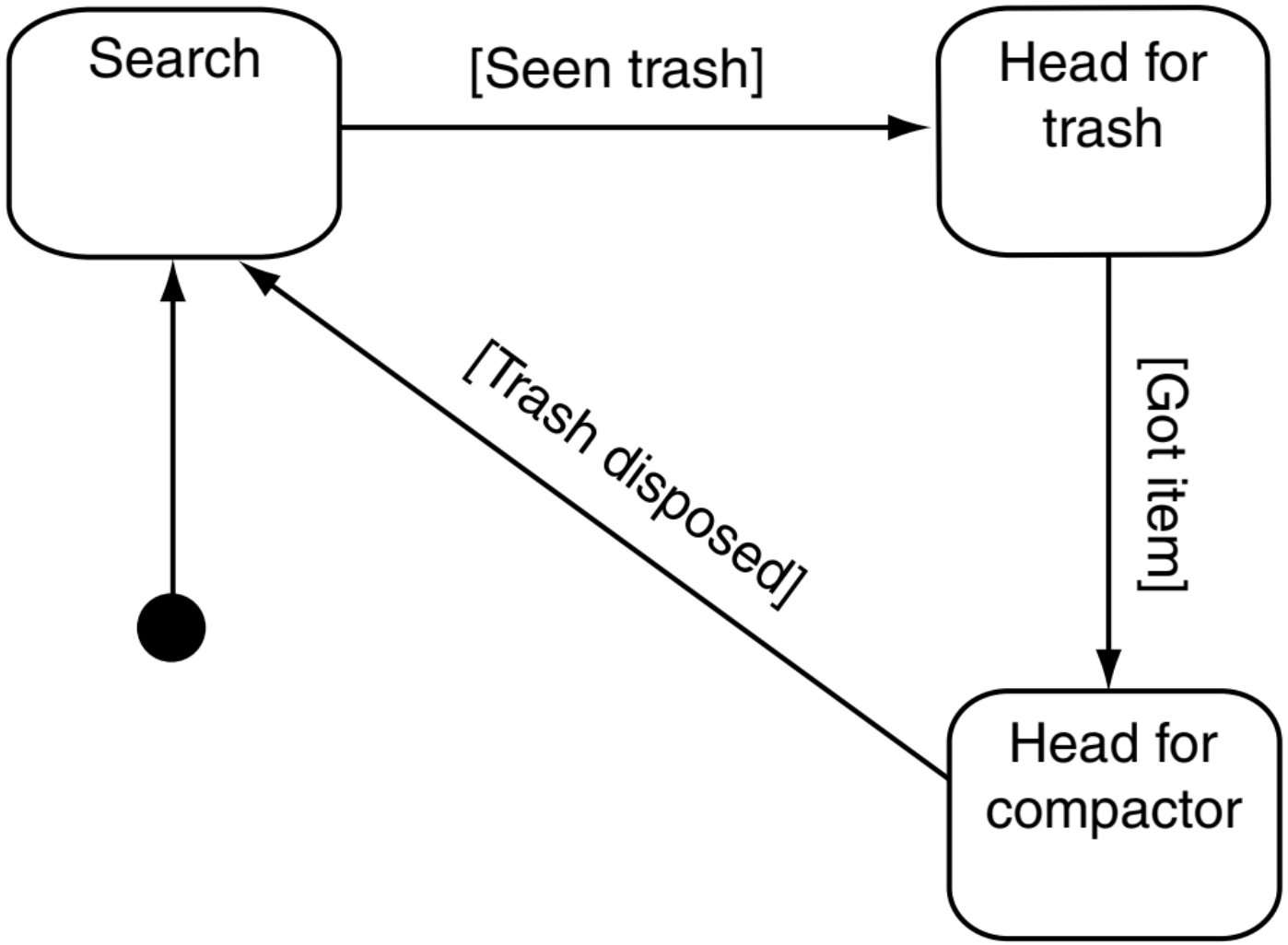
В игровых системах, равно как и в относительно небольших комплексах, представление может быть неявным – определяем множество состояний, и переходы.

Оптимизация подразумевает «упрощение» автомата – построение минимального или редукция к минимальному, эквивалентному с точки зрения функционирования (функции конечного состояния эквивалентны). Методы – алгоритмы Хопкрофта, Бжозовского и другие.

Для моделирования систем поведения ни способ представления, ни оптимизация особой роли не играют.

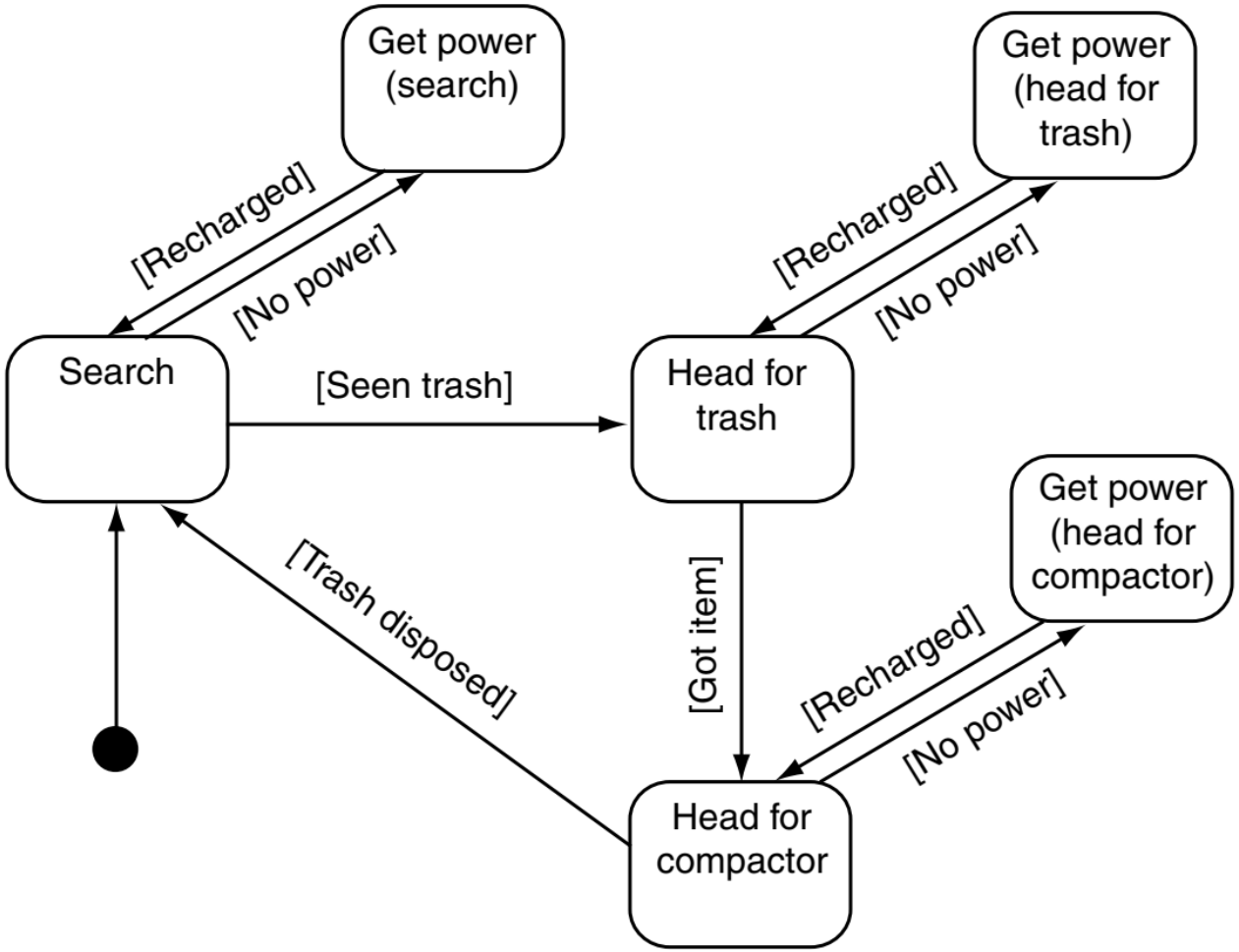
В игровом AI существенной проблемой FSA является сложность обучения (для большинства задач).

Пример


Search
[Seen trash]
Head for trash
[Got item]
Head for compactor
[Trash disposed]

Hierarchical FSM

Некоторые аспекты поведения может быть сложно реализовать – так называемые «Alarm behaviors» являются неплохим примером.

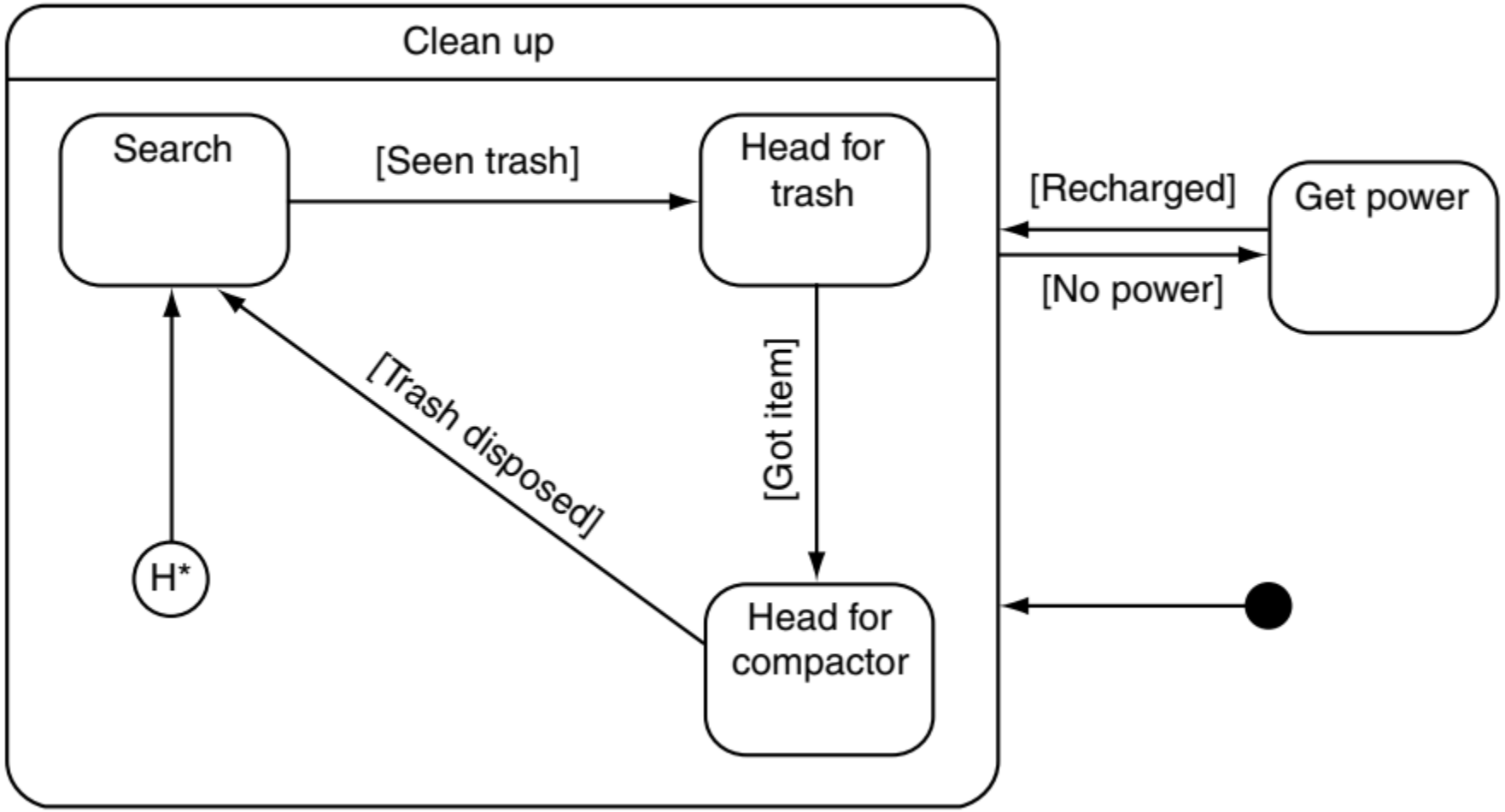


Hierarchical FSM

Решением является реализация многоуровневого автомата:

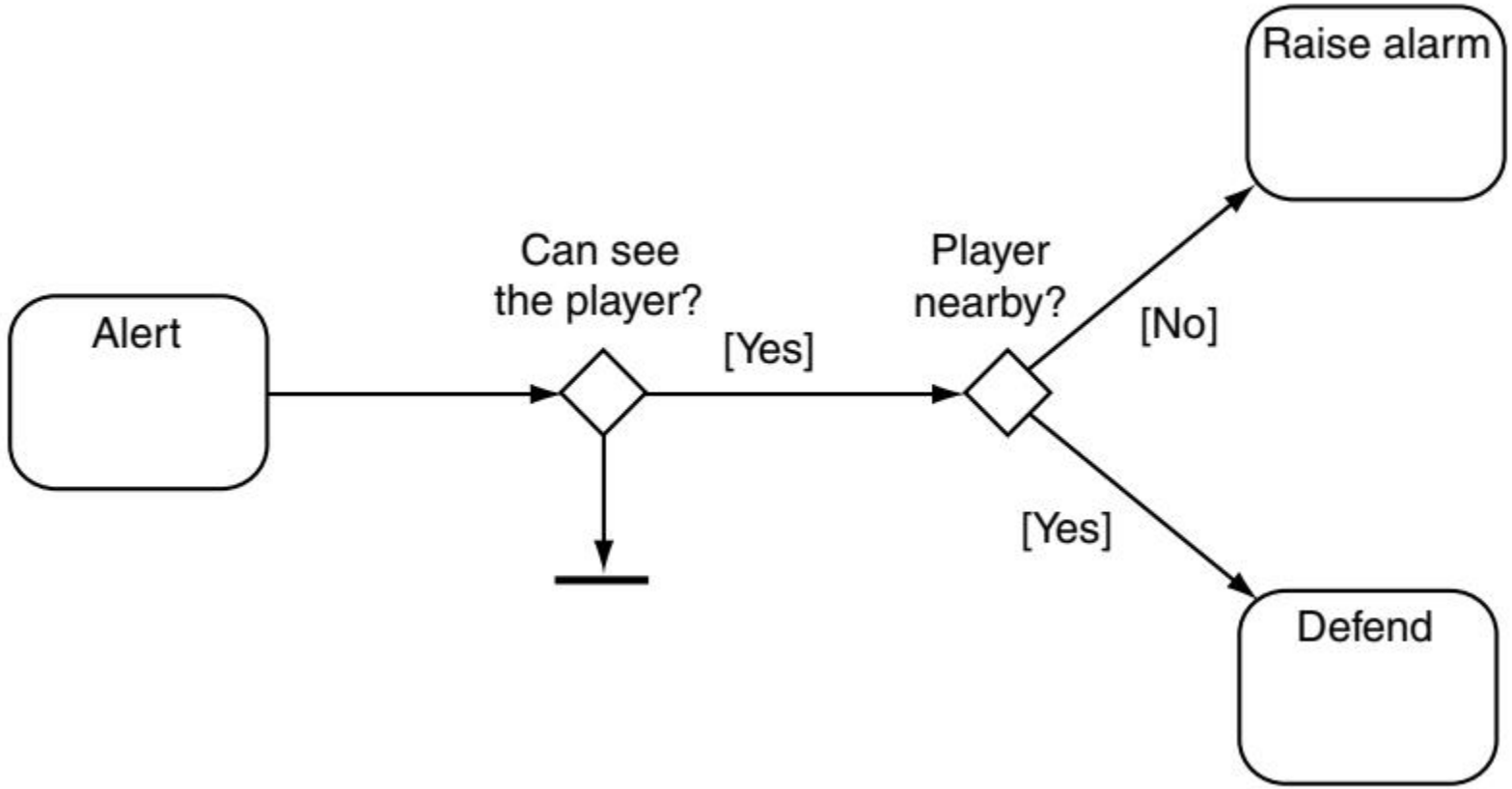
- FSM могут быть вложенными, в некоторых состояниях возможен переход во вложенные FSM;
- Получаем эффект «комбинированного состояния», при котором общее состояние системы является комбинацией нескольких состояний;
- Для каждого «вложенного» состояния необходимо указывать начальную точку входа – состояние, в которое попадаем изначально, а в дальнейшем текущая точка может запоминаться, возврат происходит в эту точку;
- Можно реализовывать переходы «между уровнями», либо только внутри иерархии.

Пример HFSM


Clean up
Search
[Seen trash]
Head for trash
[Recharged]
Get power
[No power]
[Got item]
[Trash disposed]
H*
Head for compactor

Комбинирование Decision-Trees и FSM

Довольно удобно реализуется,



Behavior Trees

Похожи на HFSM, но основной строительный блок – задачи (*task*).

В общем случае задача – некоторый код, получающий определённое время на своё выполнение, и после возвращающий статус (успешно или нет).

Виды задач могут быть следующие:

– Conditions (условия). Просто проверка на выполнение некоторого условия, возможно, довольно сложного – близость к противнику, наличие или отсутствие чего-либо, и тому подобное. Листья.

– Actions (действия). Любые действия – сделать что-либо агенту или как-либо изменить состояние мира. Листья.

– Compositions – композиции, внутренние вершины. Могут иметь дочерними другие вершины композиции. Различаются по типу, например Selector и Sequence.

Behavior Trees