Искусственный интеллект в играх

Искусственный интеллект в играх используется, чтобы реализовать поведение ботов

Для этого используются такие методы, как:

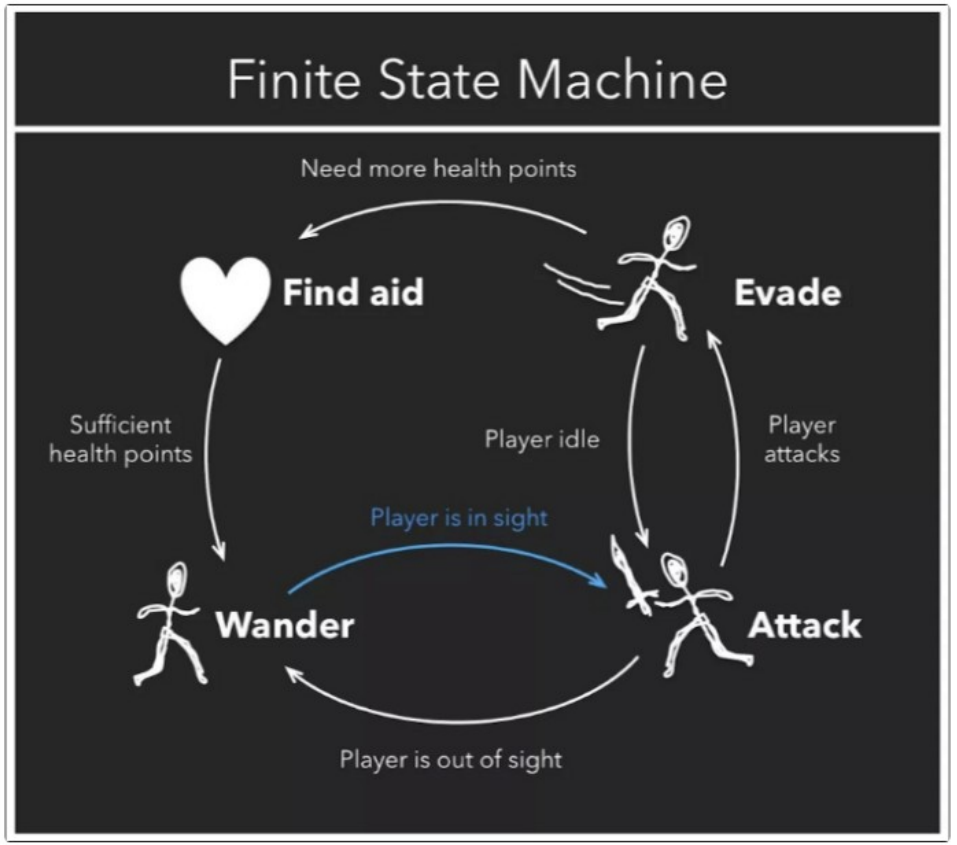
- конечные автоматы (Finite State Machine)
- метод Монте-Карло для поиска по дереву (Monte Carlo Tree Search)
- нейросетевые алгоритмы

Конечные автоматы

- Набор состояний, в которых бот может находиться

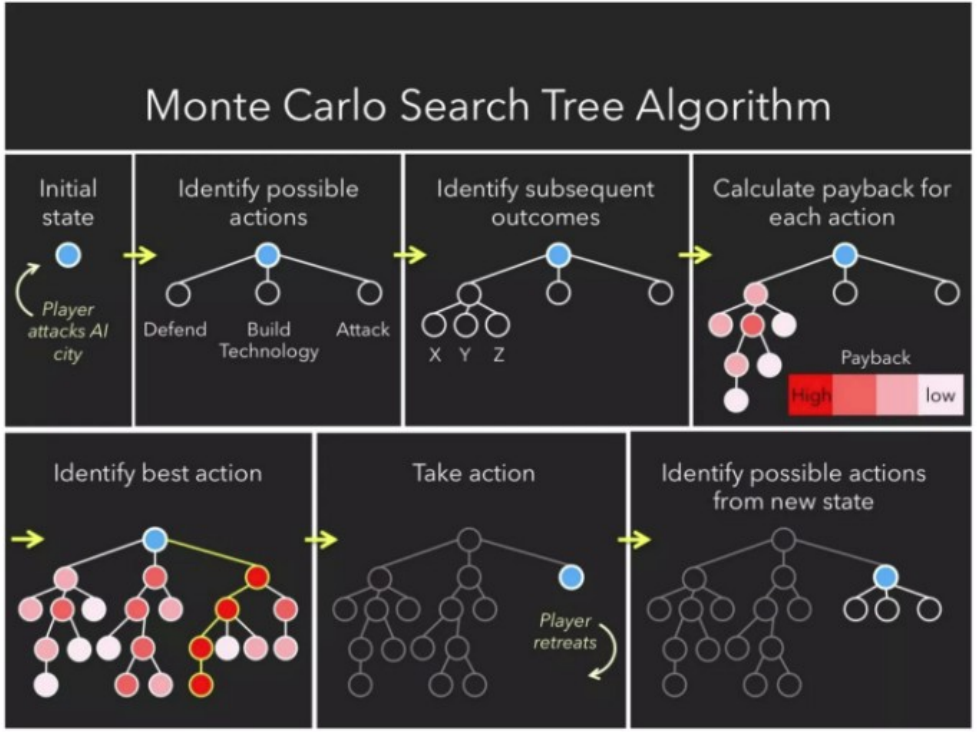
- Набор действий в каждом из состояний

- Переходы между состояниями



Метод Монте-Карло для поиска по дереву

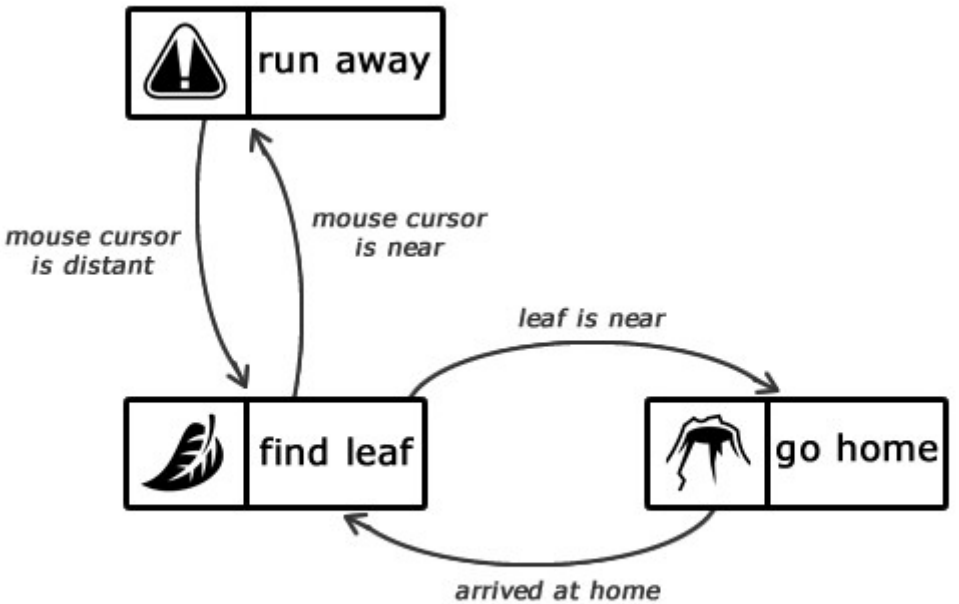
Если нужно планировать действия на несколько шагов вперед



Конечный автомат. Планирование состояний и их переходов

Реализация конечного автомата начинается с выявления его состояний и переходов между ними. Представьте себе конечный автомат, описывающий действия муравья, несущего листья в муравейник.

Отправной точкой является состояние «**find leaf**», которое остается активным до тех пор, пока муравей не найдет лист. Когда это произойдет, то состояние сменится на «**go home**». Это же состояние останется активным, пока наш муравей не доберется до муравейника. После этого состояние вновь меняется на «**find leaf**».

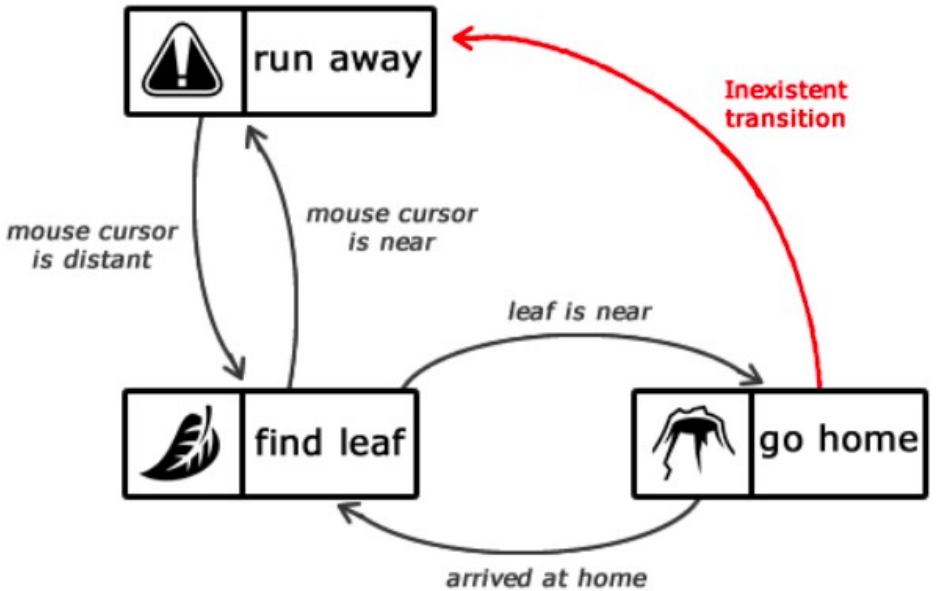


Описание состояний интеллекта муравья

Конечный автомат. Планирование состояний и их переходов

Если состояние «**find leaf**» активно, но курсор мыши находится рядом с муравьем, то состояние меняется на «**run away**». Как только муравей будет в достаточно безопасном расстоянии от курсора мыши, состояние вновь сменится на «**find leaf**».

Обратите внимание на то, что при направлении домой или из дома муравей не будет бояться курсора мыши. Почему? А потому что нет соответствующего перехода.



Описание состояний интеллекта муравья. Обратите внимание на отсутствие перехода между «run away» и «go home»

Реализация простого конечного автомата

Конечный автомат можно реализовать при помощи одного класса. Назовем его **FSM**.
Идея состоит в том, чтобы реализовать каждое состояние как функцию. Также будем использовать свойство **activeState** для определения активного состояния.

perform_action — выполнение действий для данного состояния, будет вызываться при каждом обновлении кадра игры. **activeState** — переменная типа States — активное состояние.

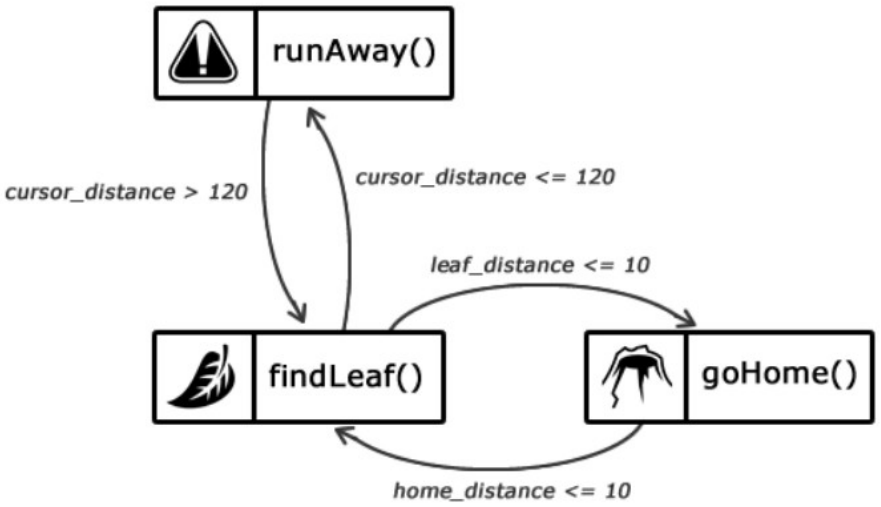
Метод **update()** класса **FSM** должен вызываться каждый кадр игры. А он, в свою очередь, будет вызывать **perform_action** того состояния, которое в данный момент является активным.

Метод **setState()** будет задавать новое активное состояние.

```
enum States
{
  FindLeaf,
  GoHome,
  RunAway
};

class FSM
{
    States activeState;
public:
    FSM() {}

    void setState(States state)
    {
        activeState = state;
    }

    void update(Ant* ant, GameValues* g)
    {
        perform_action(activeState, ant, g);
    }
};
```

Использование конечного автомата

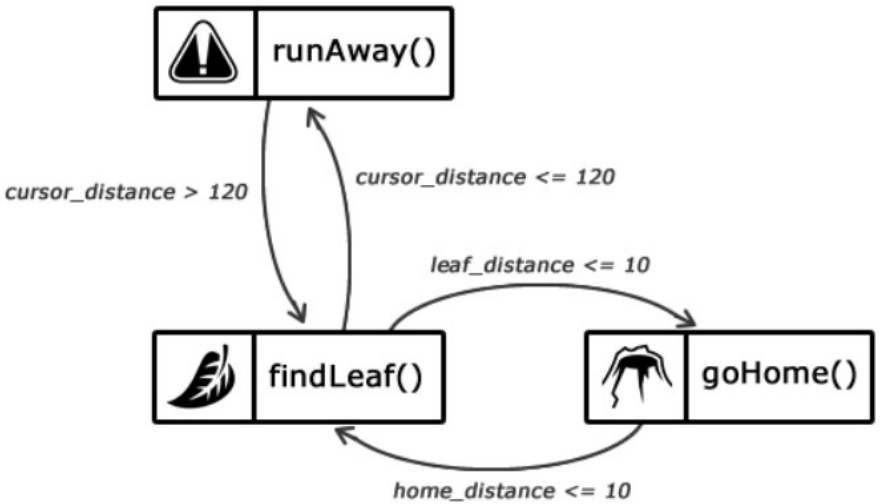
```
void findLeaf(Ant *ant, GameValues* g)
{
    array<float, 2> antPos = ant->getPos();
    // скорость - в направлении листа
    ant->setVelocity(g->leafPos - antPos);
    // Муравей только что подобрал листок, время возвращаться домой
    if (distance(antPos - g->leafPos) <= 10) {
        ant->setFSMState(States::GoHome);
    }
    // Курсор находится рядом, меняем состояние автомата на RunAway
    if (distance(antPos - g->mousePos) <= 30) {
        ant->setFSMState(States::RunAway);
    }
}

void goHome(Ant *ant, GameValues* g)
{
    array<float, 2> antPos = ant->getPos();
    // скорость - в направлении дома
    ant->setVelocity(g->homePos - antPos);
    // Муравей уже дома. Пора искать новый лист.
    if (distance(antPos - g->homePos) <= 10) {
        ant->setFSMState(States::FindLeaf);
    }
}

void runAway(Ant *ant, GameValues* g)
{
    // Перемещает муравья подальше от курсора
    array<float, 2> antPos = ant->getPos();
    if (distance(antPos - g->mousePos) > 30) {
        ant->setFSMState(States::FindLeaf);
    }
}
```



Описание состояний интеллекта муравья, сосредоточенное на коде

Использование конечного автомата

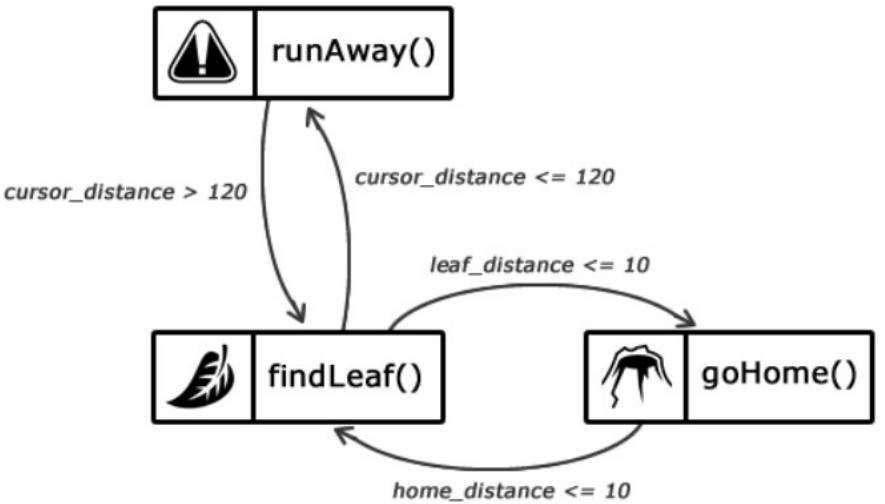
```
void perform_action(States st, GameValues* g)
{
    switch (st)
    {
        case FindLeaf:
            findLeaf(ant, lf, ms);
            break;
        case GoHome:
            goHome(ant, lf);
            break;
        case RunAway:
            runAway(ant, ms);
            break;
        default:
            cout << "state not found" << endl;
            break;
    }
}
```



Описание состояний интеллекта муравья, сосредоточенное на коде

Использование конечного автомата

```
class Ant
{
    array<float, 2> position;
    array<float, 2> velocity;
    FSM brain;
public:
    Ant(array<float, 2> position)
    {
        this->position = position;
        velocity[0] = 1.0;
        velocity[1] = 1.0;
        brain.setState(States::FindLeaf);
    }
    void setFSMState(State st)
    {
        brain.setState(st);
    }
    void update(GamrValues* g)
    {
        brain.update(this, g);
    }

};
```



Описание состояний интеллекта муравья, сосредоточенное на коде