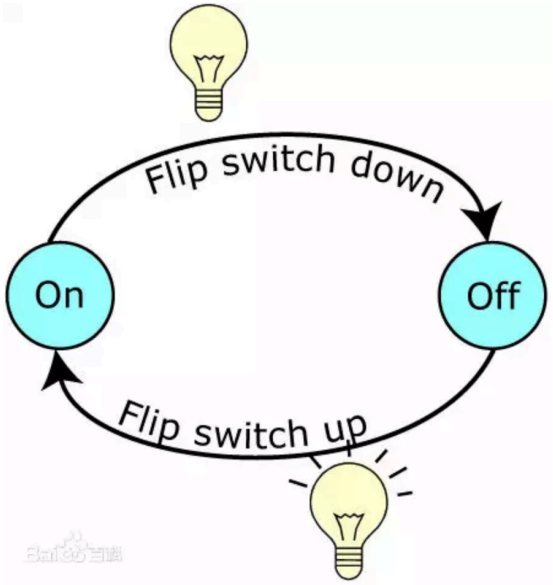
AI学习笔记--skLearn--FiniteStateMachine

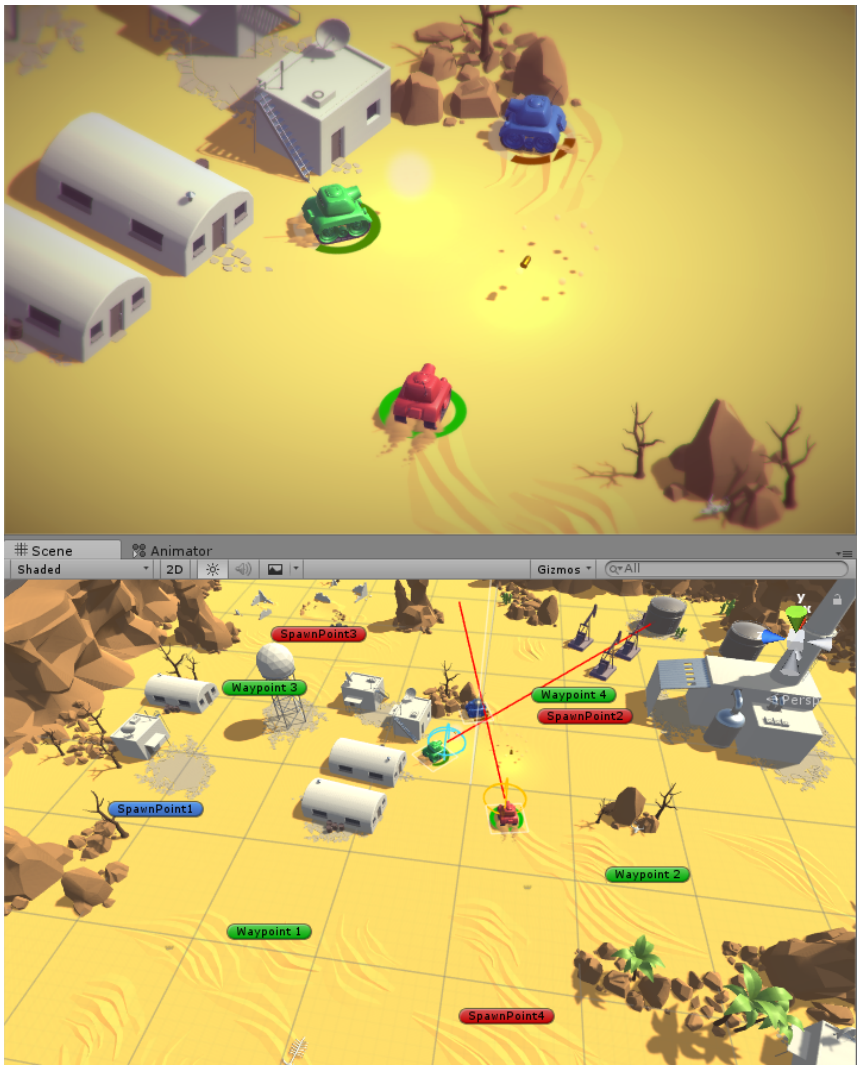
- 概述

有限状态机（FiniteStateMachine），用来表示有限个状态以及这些状态之间的转换和动作等行为的数学模型。类似下一种设计思维图：

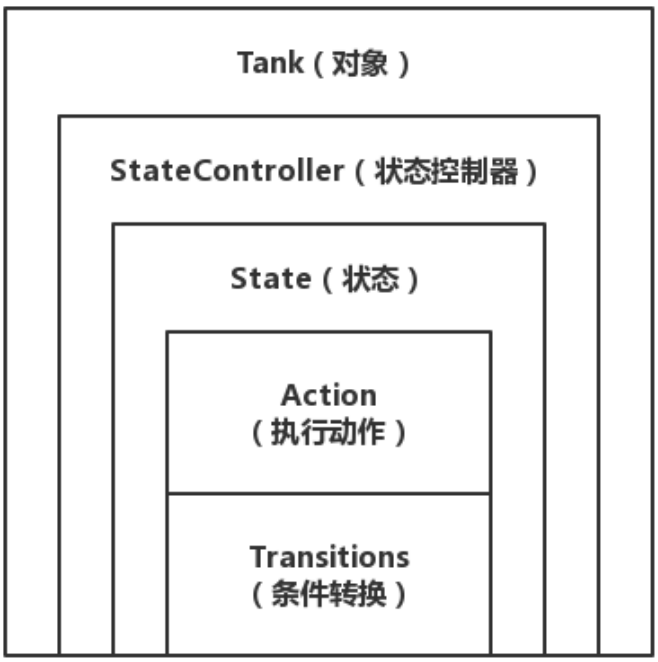


有限状态机的使用常常出现在游戏的精灵AI当中。我们举例来说一下这个数学模型，假如一款AI坦克游戏，坦克大战。坦克默认状态是巡逻（状态），在发现有敌人时（条件），转换成追杀（状态），在敌人逃脱或死掉了之后（条件），又变成巡逻（状态）。

游戏场景：绿色标签为巡逻点。蓝色标签为玩家出生点。红色为AI出生点。（如图玩家被两个AI追杀。）

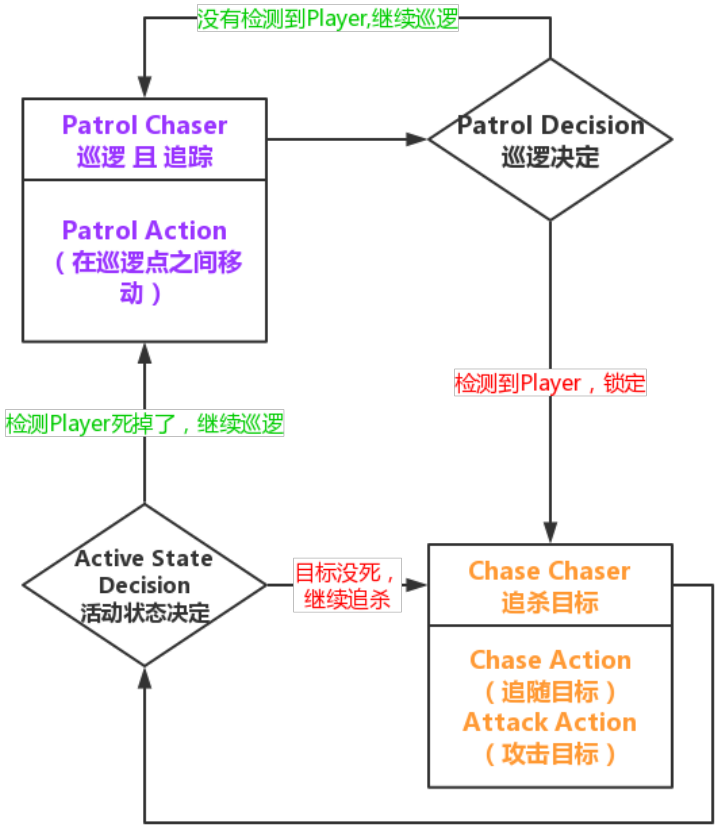


对于每个AI对象来说，都有一个状态控制器的组件，内部包含了本对象的当前状态，并且状态包含需要执行的动作和转换需要满足的硬性条件等成员。总的来说，一个FSM包含有以下几个成员：

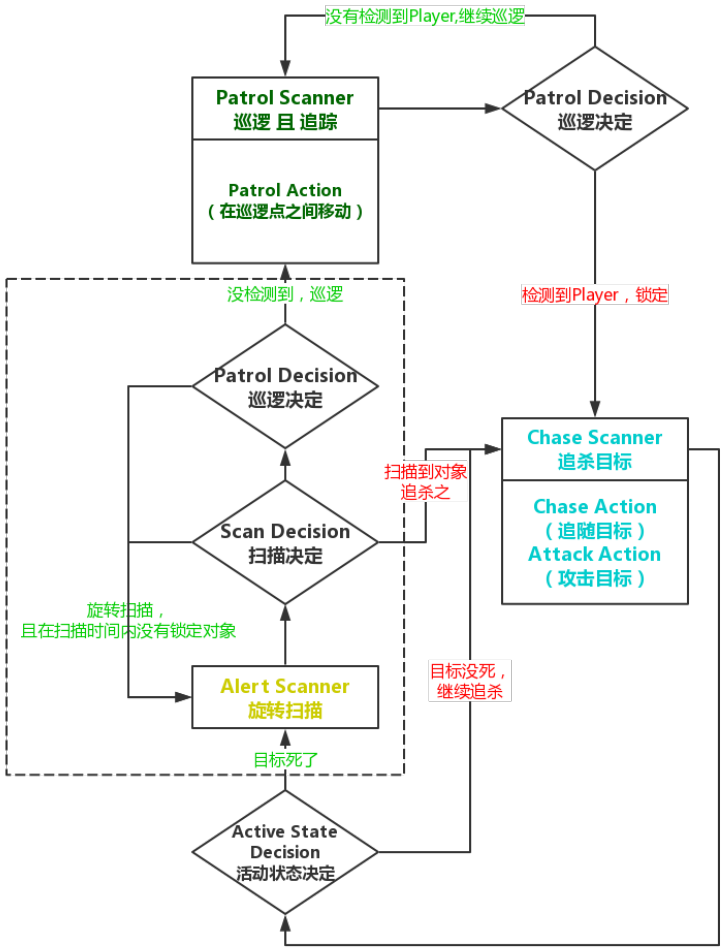


对于每个精灵来说，都具有以下几种逻辑结果：

- 第一种，巡逻->发现目标->追杀->目标死亡->巡逻。



第二种，巡逻->发现目标->追杀->目标死亡->旋转扫描目标->｛两种情况：1. 发现目标，回到第二步。2. 超过扫描时间，回到第一步（巡逻）。（颜色配合下图样例使用。）



有限状态机的设计模式，使得每个精灵、对象都有自己的一套逻辑系统，依据不同的状态，决策树有不同的执行逻辑。大体上的设计思路即使如此。

- Python Demo
  Main .py

```
# coding=utf-8
from statemachinesa import StateMachine

# 有限状态集合
positive_adjectives = ["great", "super", "fun", "entertaining",
"easy"]
negative_adjectives = ["boring", "difficult", "ugly", "bad"]


# 自定义状态转变函数
def start_transitions(txt):
    # 过指定分隔符对字符串进行切片,默认为空格分割,参数num指定分割次数
    # 将"Python is XXX"语句分割为"Python"和之后的"is XXX"
```

```
    splitted_txt = txt.split(None, 1)
    word, txt = splitted_txt if len(splitted_txt) > 1 else (txt, "")
    if word == "Python":
        newState = "Python_state"   # 如果第一个词是Python则可转换到"Python状
态"
    else:
        newState = "error_state"   # 如果第一个词不是Python则进入终止状态
    return (newState, txt)  # 返回新状态和余下的语句txt


def python_state_transitions(txt):
    splitted_txt = txt.split(None, 1)
    word, txt = splitted_txt if len(splitted_txt) > 1 else (txt, "")
    if word == "is":
        newState = "is_state"
    else:
        newState = "error_state"
    return (newState, txt)


def is_state_transitions(txt):
    splitted_txt = txt.split(None, 1)
    word, txt = splitted_txt if len(splitted_txt) > 1 else (txt, "")
    if word == "not":
        newState = "not_state"
    elif word in positive_adjectives:
        newState = "pos_state"
    elif word in negative_adjectives:
        newState = "neg_state"
    else:
        newState = "error_state"
    return (newState, txt)


def not_state_transitions(txt):
    splitted_txt = txt.split(None, 1)
    word, txt = splitted_txt if len(splitted_txt) > 1 else (txt, "")
    if word in positive_adjectives:
        newState = "neg_state"
    elif word in negative_adjectives:
        newState = "pos_state"
    else:
        newState = "error_state"
    return (newState, txt)


if __name__ == "__main__":
    m = StateMachine()
    m.add_state("Start", start_transitions)  # 添加初始状态
    m.add_state("Python_state", python_state_transitions)
    m.add_state("is_state", is_state_transitions)
```

```
    m.add_state("not_state", not_state_transitions)
    m.add_state("neg_state", None, end_state=1)  # 添加最终状态
    m.add_state("pos_state", None, end_state=1)
    m.add_state("error_state", None, end_state=1)

    m.set_start("Start")  # 设置开始状态
    m.run("Python is great")
    m.run("Python is not fun")
    m.run("Perl is ugly")
    m.run("Pythoniseasy")
```

函数处理，statemachines.py

```
    # coding=utf-8

    class StateMachine:
        def __init__(self):
            self.handlers = {}  # 状态转移函数字典
            self.startState = None  # 初始状态
            self.endStates = []  # 最终状态集合

        # 参数name为状态名,handler为状态转移函数,end_state表明是否为最终状态
        def add_state(self, name, handler, end_state=0):
            name = name.upper()  # 转换为大写
            self.handlers[name] = handler
            if end_state:
                self.endStates.append(name)

        def set_start(self, name):
            self.startState = name.upper()

        def run(self, cargo):
            try:
                handler = self.handlers[self.startState]
            except:
                print("must call .set_start() before .run()")
                # raise InitializationError("must call .set_start() before
.run()")
            if not self.endStates:
                print("at least one state must be an end_state")
                # raise InitializationError("at least one state must be an
end_state")

            # 从Start状态开始进行处理
            while True:
                (newState, cargo) = handler(cargo)  # 经过状态转移函数变换到新状
态
                if newState.upper() in self.endStates:  # 如果跳到终止状态,则
打印状态并结束循环
                    print("reached ", newState)
                    break
```

```
                else:  # 否则将转移函数切换为新状态下的转移函数
                    handler = self.handlers[newState.upper()]
```

结果：

```
('reached ', 'pos_state')
('reached ', 'neg_state')
('reached ', 'error_state')
('reached ', 'error_state')
```

测试用例以输入字符串作为结果导向，输入对应的训练关键词给决策器，决策器处理完结果输出。

。